Technical Report

CMU/SEI-88-TR-9
ESD-TR-88-010

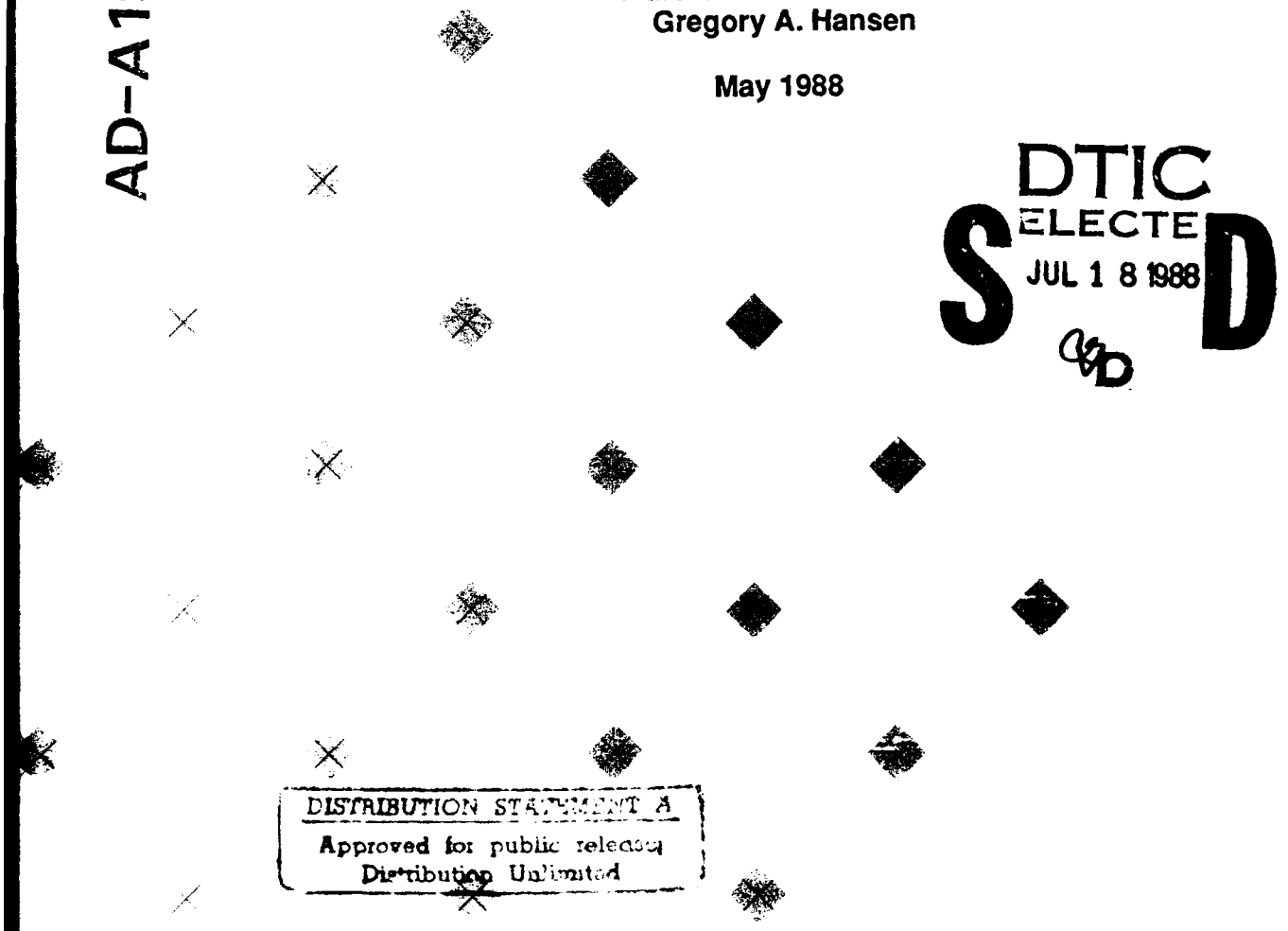
(2)

Carnegie Mellon University
Software Engineering Institute

Software Process Modeling

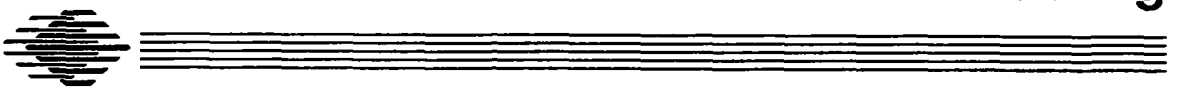
Marc I. Kellner
Gregory A. Hansen

May 1988



Technical Report
CMU/SEI-88-TR-9
ESD/TR-88-010
May 1988

Software Process Modeling



Marc I. Kellner
Gregory A. Hansen

Post Deployment Software Support
Information Management Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

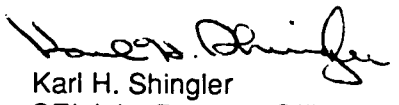
SEI Joint Program Office
ESD/XRS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER



Karl H. Shingler
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

Table of Contents

List of Figures

Software Process Modeling

Abstract:

The Software Engineering Institute (SEI), located in Pittsburgh, Pennsylvania, is a federally funded research and development center operated by Carnegie Mellon University under contract to the Department of Defense. An SEI objective is to provide leadership in software engineering and in the transition of new software engineering technology into practice. This paper discusses a software process modeling case study conducted at the SEI.

1. Introduction

This paper discusses the topic of software process modeling, a means of reasoning about the processes used to develop and maintain software. Although this term is beginning to come into common use, its meaning varies widely. For the purposes of this paper, software process modeling is defined as a methodology that encompasses a representation approach, comprehensive analysis capabilities, and the capability to make predictions regarding the effects of changes to a process.

Process modeling of this type was explored at the Software Engineering Institute (SEI) during the execution of the Post Deployment Software Support (PDSS) Information Management Project, which focused on improving the process used by the Air Force to modify *Technical Orders* (TOs) to correspond to software changes to a weapon system. Early in the project, it became clear that this process was complicated as it involved a number of organizational subunits. It also became clear that there was a considerable shortage of personnel who understood the full process or who could see how their portion fit into an overall context. Believing that one needs to understand a process before attempting to improve it, project members concluded that an important aspect of the project effort would be to describe and analyze the organizational process employed.

This paper reports the approach we have taken to software process modeling and summarizes our experiences on the PDSS Project. The paper is structured as follows:

- Chapter 1 - Introduction.
- Chapter 2 - Overview of the objectives and capabilities of software process modeling.
- Chapter 3 - Overview of the PDSS Information Management Project, describing the context for our modeling experience.
- Chapter 4 - Details and examples of the modeling approach used on the PDSS Project.
- Chapter 5 - Outcomes and results of those modeling efforts.
- Chapter 6 - Lessons learned from this effort, and a list of the capabilities required for successful software process modeling.
- Chapter 7 - Conclusions and directions for future work.

2. Overview of Software Process Modeling

2.1. Major Objectives of Software Process Modeling

The PDSS Information Management Project is part of the Software Process Program at the SEI. A basic premise guiding work in this program is that the quality of a software product is largely determined by the quality of the process used to develop and maintain it. Thus, product improvements can be effectively achieved by improving the software life cycle processes; consequently, a primary goal of software process work is to facilitate and enhance the deliberate and planned evolution of a process toward greater effectiveness, efficiency and reliability. This process evolution leads directly to lasting improvements to the corresponding software products. It is more important today than ever before to model and analyze the software process because the advent of new technologies is forcing managers and developers to decide how to best utilize these technologies. Historically, those decisions have been made intuitively, with little empirical data to substantiate promises of gains in productivity and product quality. However, software process modeling supports the goal of planned process improvement by providing a mechanism for:

- Recording and understanding the baseline process.
- Evaluating, communicating, and promulgating process improvements.

At a more detailed level, we have identified four primary objectives for the development of models of the software process:

1. Enable effective communications regarding the process.
2. Facilitate reuse of the process.
3. Support evolution of the process.
4. Facilitate management of the process.

The first objective focuses on effectively communicating the description of a process to others, such as workers, managers, and customers [9]. Process models are especially useful for sharing knowledge and expertise; for example, process descriptions can be used for training purposes. It is crucial that personnel be able to view a process at different levels of abstraction; i.e., they should be able to start understanding the process from the top down, from the bottom up, or even from the middle working either way.

Reuse, the second objective, enables a specific software process to be instantiated and executed in a reliably repeatable fashion across multiple software projects. This reuse may occur within a single organization or across organizations. Osterweil has suggested that ". . . the most important benefit of process [modeling] is that it offers the hope that software processes themselves can be reused" [9].

The third objective is to support the evolution of the process, through (1) serving as a storehouse for modifications, lessons learned, and tailoring; and (2) analyzing the effectiveness of changes in a laboratory or simulated environment before actually implementing them. Successful tailoring decisions should be formalized and stored as part of the model, so that they can be consistently applied in the future. In addition, deliberate evolution is greatly enhanced by an ability to qualitatively and quantitatively examine potential process modifications in a simulation mode before trying them in practice.

The final objective is to facilitate effective planning, control, and operational management of software processes. This is accomplished through increased understanding of the process, conformity to process definitions, quantitative simulation and analysis capabilities, training, and other activities.

2.2. Primary Capabilities Required for Software Process Modeling

In order to accomplish the above objectives, software process modeling must possess capabilities in three major categories:

- representation
- comprehensive analysis
- forecasting

We anticipate that these requirements will be met most effectively through an automated approach.

A powerful representation formalism is required to cope with the complexities of actual organizational processes. We believe that it is important to be able to represent descriptions of the current ("as-is") process, prescriptions of a desired future ("to-be") process, and restrictions imposed by regulations and standards.

Comprehensive analysis capabilities must include a wide variety of tests in the areas of consistency, completeness, and correctness. These are critical in determining the validity of the model itself, and of the real world process which the model represents.

Forecasting capabilities include both qualitative and quantitative aspects. Qualitative examinations look at the behavior of the process in response to various events and circumstances. Quantitative examinations extend these to predict numerical outcomes along dimensions such as time-to-completion, manpower requirements, or quality measures. We see these capabilities being provided through simulation that is tightly integrated with the model representation and analysis features. This vehicle is ideal for answering "what if" questions about such activities as procedural changes and technology insertion.

3. PDSS Project Overview

3.1. Project Purpose

Post deployment software support (PDSS) has been defined as

> "... the sum of all activities required to ensure that, during the production/deployment phase of a mission-critical computer system's life, the implemented and fielded software/system continues to support its original missions, and subsequent mission modifications and product improvements" [7].

PDSS, therefore, includes not only software "maintenance," but also the activities required for overall system support.

In exploring PDSS activities and concerns across the Services, SEI personnel discovered that many support organizations experienced difficulties modifying and delivering user-oriented technical documentation corresponding to new releases of software systems. Since the management of the documentation modification process presents a significant challenge and directly relates to the availability of mission-critical systems, the SEI initiated the PDSS Information Management Project.

The purpose of this project was to demonstrate the feasibility of dramatically reducing the delay between completion of software changes and distribution of the corresponding modifications to "user" documentation. This time lag has repeatedly resulted in delays of several months in fielding a complete software change package.

Our strategy on this project was to focus on a specific representative case, in particular, the Operational Flight Program (OFP) for the F-16 A/B aircraft. The OFP is the avionics software that helps fly the plane and manage its defensive systems. PDSS activities for this airplane are managed by the Ogden Air Logistics Center (OO-ALC) at Hill Air Force Base in Ogden, Utah. In the Air Force, the user documents are called Technical Orders (TOs); for the F-16, TOs are primarily directed at pilots and maintenance personnel.

Our strategy included:

- Examining the TO modification process and recommending improvements to that process.
- Conducting a pilot study applying advanced document production technology to a sample TO for the current OFP block change.
- Assessing the impact of this technology at Ogden and providing empirical data regarding productivity enhancements and conversion costs.

It is useful to have a perspective on the scope and magnitude of the effort to produce TO modifications, which has been a major bottleneck in the overall process of developing and fielding software upgrades for many weapon systems, including the F-16. The USAF has approximately 126,000 active TOs, spanning about 11,000,000 pages. Approximately one

million change pages were processed in the 1984 fiscal year. Of course, only a portion of these were due to software changes. Turning to the F-16, there are about 141,000 pages of aircraft TOs unique to the F-16 A/B. General Dynamics, the prime contractor for the F-16, reported that a "typical F-16 A/B Operational Flight Program (OFP) block change" affected:

- 80 aircraft TOs (total of 3,817 pages changed)
- 24 commodity TOs (total of 83 pages changed)
- 24 time compliance TOs (total of 51 pages created)

Many of the affected TOs are modified in relatively minor ways. For example, many maintenance and job guide TOs involve changes in part numbers to correspond to the software change; and many maintenance and operational TOs, as well as job guides and checklists, must be modified because the OFP software identification number changes. (This number appears in numerous diagrams depicting the expected display on the "Power-On Panel" when the system is first turned on.) Although these changes are relatively easy to make once identified, their identification from 141,000 pages of manuals can be quite challenging. On the other hand, a relatively small number of TOs undergo far more substantial modifications. These include several operational manuals, since many software changes affect the "user interface". Our pilot study worked with one of these highly volatile operational manuals as the most difficult case.

3.2. Synopsis of the TO Modification Process

The PDSS Project's activities involved understanding, documenting, describing, and analyzing the process currently used at OO-ALC to produce and distribute TO modifications that corresponded to the F-16 A/B OFP avionics software block changes. We are convinced that such an exercise is a necessary precursor to successful process improvement efforts. This process description provided a base from which project members formulated recommendations about possible improvements and technological enhancements to streamline the process. Furthermore, we expected this description to be helpful to those involved in any phase of the process because it clarifies the entire process and enhances their appreciation for the part their role plays in its successful execution.

Information was gathered primarily through interviews with the personnel directly involved in executing the process steps. These interviews were supplemented with information obtained from those involved in managing the process, and with information gleaned from regulations bearing on the process steps. Our intent was to model the process that actually occurred for the most recent OFP block change—called "15S1."

To help us organize and synthesize the information, we first developed a structured narrative description of the major process steps. An example of one of these structured narrative descriptions appears in Figure A-1. To clarify the narrative descriptions of the process flow, we developed the diagram which is presented as Figure A-2.

The structure of Figure A-2 loosely follows that of a data flow diagram, a technique widely used in systems analysis work. The fundamental focus of this approach is to trace the flow of data or information through the system. The lines with arrowheads in the figure represent the data flows between activities and data stores. Data stores, repositories where data are stored for future reference, are represented in the diagram by short, wide rectangles, open on the right side. There are four stores shown in the figure, numbered S1, S2, S3, and S4. One of these, S2 (TO Libraries), is shown twice for convenience in drawing the data flows. The seven rectangles with rounded corners represent the basic activities carried out in the system. In this case, these activities occur in a basically sequential fashion. Finally, a square is used to depict external activities, personnel, etc. This symbol is used in Figure A-2 to represent Technical Order Distribution Offices (TODOs), the recipients of the modified TOs.

Because a basic understanding of the process will be useful in appreciating the examples of modeling approaches and specific model components to be discussed later, a synopsis of the process will now be presented. Additional details of the process are documented in [1, 2].

The first major activity in the TO modification process is labeled in Figure A-2 as "1.0 Identify and Draft Modifications to Affected TOs." This activity is performed primarily by software engineers (working in an organizational unit designated MMEC) who have been involved in the design and testing of the OFP changes. Accomplished in a purely manual fashion, the activity involves thumbing through thousands of pages of TOs looking for text and diagrams that need to be modified. The draft changes are then marked in red pen (red-lined). When all draft changes have been made, the entire package of all red-lined TOs is then passed to another organizational unit (designated MMAR) for review and ultimate approval—"2.0 Review and Dratt Modifications to Affected TOs" in the figure. TOs may be returned to MMEC if they need to be reworked before approval. After approval, an AFLC Form 252, which details the changes to be made, is prepared and signed for each affected TO. These forms are held until the software changes have been validated and verified; then they are forwarded to another organizational unit as a single package.

The package of approved TO changes authorizes the MMEDT office to compose and typeset reproducible copy of the formal documents that detail the changes and are issued to the field. This activity is depicted in Figure A-2 as "3.0 Prepare TO Modifications for Printing." The actual production work of composing, drafting graphics, typesetting, and merging text and graphics may be performed by OO-ALC personnel or by a local overflow contractor. Normally, one of two types of TO modifications are prepared, depending on whether the affected TO is physically maintained at OO-ALC or General Dynamics. In the latter case, which applies to most affected TOs, an operational supplement (op sup) is normally developed by MMEDT as the document form distributed to the field. In the former case, change pages are generally produced. As the changes for each TO are completed, a reproducible copy of each TO modification document is forwarded to another organizational unit (DARA) for printing, along with instructions on how many copies to print and an indication of the turnaround time allowed for printing.

The fourth activity shown in Figure A-2 is labeled "4.0 Printing." The purpose of this activity is to print the required number of copies of each op sup or group of TO change pages within the allowed turnaround time. The printed documents are then forwarded to the TO warehouse for distribution.

The distribution activity, "5.0 Distribution," is managed by the TO Distribution Control Office (TODCO), a MMEDT function. The TO documents are packaged, addressed, metered, and mailed. This is the end of the process for those TOs for which change pages were produced.

Additional steps are applied for the majority of the TOs—those for which op sups were produced. Copies of the op sups are held on file at MMEDT and the next activity begins—"6.0 Prepare TO Change Order to Contractor." Its purpose is to eventually incorporate TO changes published as op sups into actual TO change pages. Since General Dynamics is contracted to maintain these TOs, they must actually prepare the change pages. Normally, MMEDT personnel wait until the next time a change processed through OO-ALC/MMEDT is to occur. At that point, they will direct General Dynamics to incorporate the new change, as well as those previously issued in the op sup, into formal change pages. In general, this new change is entirely unrelated to the previous software-related changes. It should also be noted that this activity proceeds independently for each affected TO, so some op sups are incorporated into change pages before others are. The directive to General Dynamics is issued as a TO Change Order.

Upon receipt of this order, General Dynamics performs the final activity shown in Figure A-2, "7.0 Prepare, Print, and Distribute TO Change Pages." The print shop usually mails the documents directly.

4. Modeling Approaches

4.1. Initial Approaches and Alternatives

When we first developed process documentation in the form of structured narrative descriptions of the various process steps, the results were rather unwieldy. Two of the most obvious shortcomings of this approach are the lack of a big picture of the overall process, and the voluminous explosion of detail that results as one resolves a major activity into its detailed components.

We then supplemented the narrative descriptions with the data flow diagram (DFD) illustrated in Figure A-2. Using this technique we can convey in one diagram an overall idea of the workings of the TO modification process. We have employed the diagram effectively in our reports and presentations, and the technique is widely used in systems analysis and design work. However, data flow diagrams provide only limited information. They convey only a *functional* view of a process; i.e., they describe only what is being done and what data is flowing. The diagrams provide no information about the timing of the process steps. Certainly, one can make precedence inferences from a DFD, but it is not clear where parallelism occurs (versus sequential processing), what actually triggers the commencement or termination of a step, etc. For example, the diagram does not make it clear that the major input into Activity 3.0 is a monolithic package of all TO changes, whereas the output is a reproducible copy for each TO, each of which is transferred as it is completed individually. Nor is it clear that document preparation (Activity 3.0) occurs in parallel with printing (Activity 4.0) because the modifications for some TOs are being printed while others are still in preparation. Hence, it is also unclear that printing commences when the modifications for the first TO are ready. In addition, DFDs say nothing about how the functionality is actually implemented in the organization: what organizational unit performs the work; how information flows (e.g., by e-mail, verbally, US mail). Finally, our DFD was a simple drawing; it had no processing power behind it and could not be automatically analyzed.

As a result of these shortcomings, we searched for a more powerful approach to software process modeling. We decided to select a methodology that had existing automated tools to support its application, since the lack of such tools would severely restrict application to simple processes only. We formulated a preliminary set of requirements and examined a number of potential modeling approaches. These included:

- Project management toolkits that included PERT and/or CPM techniques for critical path analysis.
- Simulation languages, such as SIMSCRIPT or SIMULA, which use a software program to model the behavior of the process being examined.
- Expert systems packages, some of which also had simulation capabilities (these had frequently been employed in modeling factory work flow alternatives).
- Systems analysis and design toolkits for traditional commercial MIS

applications—typically employing the Yourdon structured design approach or similar methodology.

- Approaches used by commercial organizations with large consulting practices in the area of systems analysis and design, including some firms that often deal with automating existing organizational processes.
- Toolkits for specification and analysis of real-time automated systems.

This review revealed several automated systems that provided many of the capabilities we sought. We acquired one of these systems, called STATEMATE (offered by i-Logix, Inc. of Burlington, MA), and have been using it on a trial basis to develop software process descriptions, and to analyze and simulate their behavior. This system supports a unique methodology which was originally developed to aid in the design of real-time reactive systems (e.g., avionics software). However, it appears to be quite promising as an approach for software process modeling.

4.2. Software Process Modeling with STATEMATE

This section provides an introduction to the STATEMATE approach to software process modeling, including a number of examples from our actual model of the TO modification process. No attempt is made to exhaustively cover the capabilities of STATEMATE for modeling; these are documented fully in [5, 6]. Nor will we exhaustively describe the meaning of the model details. Rather, our aim is to illustrate the "flavor" of the modeling approach we are using.

4.2.1. Orientation

STATEMATE offers model builders a set of three distinct but interrelated viewpoints with which to model a real-world process. Taken as a whole, they cover the traditional "who, what, where, when, and how" of a process. The three viewpoints are:

- Functional — what is done; represented by activity charts.
- Behavioral — when and how it is done; represented by statecharts.
- Structural — who does it and where is it done; represented by module charts.

These three viewpoints are just three ways of looking at the same process. They are different because they look at the process from different vantage points or perspectives. Nevertheless, they interrelate because they all describe the same process.

In addition, there are explicit interconnections between the views: specifically, between the functional view and the other two. The relationships among the viewpoints is illustrated in Figure A-3.

With STATEMATE, a process is described through graphics and through textual forms. The graphics are used to describe the three viewpoints as activity charts, statecharts, and module charts. The textual forms are used to describe connections between the views, formal definitions, and informal narrative descriptions.

The graphical languages utilized for the three viewpoints are quite similar. They use two major components: named boxes and directed lines. The named boxes represent activities, states, and modules, respectively, in the three types of charts. Hierarchical decomposition is represented by nesting within the same diagram. This is in contrast to most other toolkits, which use separate diagrams or windows to depict different levels of detail. Directed lines represent information flows in activity charts and module charts; they represent state transitions in statecharts. Grouping notions apply these to hierarchy.

STATEMATE runs on Digital Equipment Corporation (DEC) VAX computers under the VMS operating system. Our installation is running on a VAXStation GPX, which has a MicroVAX II CPU and a 19-inch color monitor with mouse. Both VMS and STATEMATE run in a multiple window environment on the GPX monitor.

STATEMATE provides considerable flexibility. One can start with any of the views, readily switch between the different charts, and even work on the different charts in parallel in different windows. In addition, one can readily follow a top-down approach to decomposition, a bottom-up approach, or even a middle-out approach. In the examples presented below, we will present the model by examining the statechart first, followed by the activity chart and the module chart. We will follow a top-down approach for clarity of exposition; one can actually develop and examine the model components in a highly flexible fashion.

4.2.2. Behavioral Viewpoint — Statecharts

Statecharts provide the behavioral description of the models, i.e., the when and how. Statecharts are an improved variety of state transition diagrams [3, 4]. They extend traditional state transition diagrams by allowing one to (1) decompose states in an and/or fashion, including hierarchy, (2) direct state entrances by the system's history, and (3) specify timing constraints. The major components of a statechart are simply states and transitions between states.

Transitions are labeled with a trigger and/or a set of actions. When the trigger occurs, the state transition is taken and the actions are performed. Example actions include: generate an event; set a condition; and perform a calculation. Triggers can be quite general, including an event expression and a condition expression. (A transition occurs when the event expression occurs while the condition expression is true). An example trigger, taken from our model, is

```
(INIT_REV_DONE or RE_REV_DONE) [REWORK_NEEDED]
```

INIT_REV_DONE is an atomic event that occurs the instant the initial review of red-line changes is completed. Similarly, RE_REV_DONE is an atomic event that occurs the instant a re-review (of reworked changes) is completed. If either of these events occurs, the event portion of the trigger is satisfied. REWORK_NEEDED is a condition, meaning that it is a Boolean variable with the possible values true or false. This variable indicates the outcome of the review. If it has the value true at an instant when one of the events occurs, then the entire trigger is satisfied and the corresponding state transition will be taken. To simplify the

diagrams, one can assign convenient names to complex triggers; in our model, this example is the actual definition of the defined event NEED_REVS. Figure A-4 shows the STATEMATE dictionary listing for this event, including an explanatory narrative description.

It is worth noting that the STATEMATE approach distinguishes between the instantaneous and the continuous. Instantaneous concepts take zero time; these include events, state transitions, and actions. On the other hand, continuous concepts persist for a positive time interval; these include conditions, states, activities, and data values.

Figure A-5 presents the top-level view of our statechart representing the behavior of the TO modification process. The line starting in the upper left corner and leading to the box marked IDLE is called a default transition and indicates where the system starts off. Thus, we defined the process to begin in an idle state. When the GO_AHEAD event occurs, the process moves from the idle state into the DRAFT_MODS state (drafting modifications). When the DRAFTS_DONE event occurs, the process transitions to the REVIEW_MODS state (performing the review). The process may iterate between these two states. This will happen if the NEED_REVS trigger is met (its definition was discussed above). Eventually, all TO modifications have been approved, which is signaled by the ALL_APPROVED trigger. The process then continues into the ISSUE_INTERIMS state. This process component was discovered by our team after the initial data flow diagram had been developed; therefore, it is included in our STATEMATE model but not in our earlier model or discussion.

When the event CHG_AUTH_DONE occurs (change authorizations done), the process moves to a more complex state, labeled by the tab PREPARE_PRINT. This state with the dashed line in it is called an "and-state". It has two (in this case) orthogonal components: DOC_PREP (document preparation) and PRINT (printing). The meaning of these orthogonal components is that they can occur in parallel. In other words, some TO modifications are in the state of being printed, while others are still in the document preparation stage. Previous top-level states occurred in a purely sequential fashion. The coordination details between orthogonal states are described at lower levels of detail, as will be seen in a moment.

As the work progresses, the process moves to a DISTRIBUTE state, then to another and-state labeled CONTRACTOR_CHG (for contractor changes to the TOs). Finally, when ALL_CHG_PGS_DONE triggers, the process moves to a circle containing a "T," which is a termination symbol meaning the process is complete.

Figure A-6 shows the statechart with the next level of detail revealed. Additional detail is represented by nesting. For example, INITIAL_REVIEW and REWORK_REVIEW are two substates of REVIEW_MODS from the top-level diagram. Different portions of the process can be resolved into varying levels of detail, as required. For example, DISTRIBUTE is an atomic-level, basic state with no substates. On the other hand, DRAFT_MODS has two additional levels of detail, as shown in Figure A-6 and, in finer resolution, in Figure A-7.

Returning to the issue of and-states, consider PREPARE_PRINT in Figure A-6. Its component DOC_PREP contains another and-state, PREPARING, which has as many as four simultaneous activities. The details of the PREPARING state are exploded in Figure A-8.

Synchronization between orthogonal components of an and-state is specified at lower levels of detail. As an example, consider CONTRACTOR_CHG in Figure A-6. When this and-state is first entered, both orthogonal components start in their respective IDLE substates, as indicated by the default arrows. When the external event NEXT_CHG occurs, meaning another change is needed, the process advances to the DEVELP_CHG_ORDER state, where a change order will be developed. Meanwhile, CONTRACTOR_WORK remains in IDLE. When the CHG_ORD_SENT event occurs (the change order is sent to the contractor), PREPARE_CHG_ORDR returns to its IDLE substate, and simultaneously the CONTRACTOR_WORK status moves from IDLE to DO_CHG_PGS. When the NO_WORK_WAITING event occurs, CONTRACTOR_WORK returns to IDLE. This arrangement allows the possibility of a queue of work at the contractor's shop. The details of tracking this queue are managed within the DO_CHG_PGS state. The textual form details of this state are reproduced in Figure A-9. States are allowed to have "reactions." These are defined in the same way as transition labels, with a trigger portion and an action portion; but when they are triggered, only the actions are executed, with no state transition being taken. Thus, we defined a variable NUM_IN CONT_Q to track the number of TO jobs in the contractor's queue. It is incremented when the state is entered and whenever the event CHG_ORD_SENT occurs while the process is in the DO_CHG_PGS state. Each time a TO change is finished, the event A_TO_CHG_BY_CONT is generated, which decrements the count in the queue. When the count reaches zero, the event NO_WORK_WAITING occurs, because this event is defined as

```
tr(NUM_IN_CONT_Q=0)
```

This means that the event occurs at the instant the condition in parentheses turns true.

These examples convey the "flavor" of the manner in which STATEMATE represents the behavior of a process; however, many additional capabilities are also available.

4.2.3. Functional Viewpoint — Activity Charts

Activity charts provide the functional description of the models; i.e., they show the tasks that are performed. Basically, activity charts are enhanced data flow diagrams. The major components of an activity chart are activities, possible information flow lines, and data stores. An activity chart without the information flow lines is shown in Figure A-10; the flow lines have been suppressed in the diagram to improve its readability. There are three types of activities: normal, external, and control. Normal activities are the main ones being modeled; they depict a specific (sub)function within the process, such as DRAFT_MODS or PRINT_DOCS in Figure A-10, and they can be resolved into additional detail. External activities are shown in the figure as boxes with dashed outlines, such as TODOS and XTRNL_PRNTRS. Since these are outside the scope of the process being modeled, they are not included in any detail except to recognize their existence and information flows. The

two examples noted correspond to the recipients of TO modifications (Technical Order Distribution Offices) and external print shops, respectively. Control activities correspond to statecharts. They are depicted by rectangles with rounded corners, as shown in the lower left corner of the activity chart. The name TO_CTRL_1 provides the linkage to the statechart illustrated previously. The control activity controls the normal activities which are its siblings: DRAFT_MODS, REVIEW, etc. It can start or stop a normal activity, suspend or resume it, and so forth. Finally, a data store is illustrated in the upper center of the figure, labeled TO_LIBS; this is a repository for information. Visual distinctions between these constructs are much more obvious on the screen, where color is used to distinguish among types of boxes.

There are two types of information flow lines, those for processing information and those for control information. Figure A-11 adds the information flow lines to the activity chart. Solid lines are used to depict the flow of information to be processed, such as variables, records, and lists. In the lower right corner of the diagram we see an information flow labeled TO_CHG_PGS, from the CONT_CHG activity to the TODOS external activity. Of course, this represents the delivery of completed formal change pages to a TO. An example flow of information used for control purposes appears at the bottom of the diagram; it is labeled A_TO_CHG_BY_CONT and flows from the CONT_CHG activity to the control activity (corresponding to the statechart). This item, which is an event, was briefly discussed at the conclusion of Section 4.2.

As is the case for all STATEMATE diagrams, additional levels of detail are provided by nesting more decomposition into a box. For example, the details of the DRAFT_MODS and REVIEW activities, respectively, are illustrated in Figures A-12 and A-13.

Connections between states and activities may be specified explicitly on the statechart or implicitly through the textual forms. Actions are allowed within statecharts to start, stop, suspend, and resume any activities by name. One can also use the forms to indicate a connection between a state and an activity. The "throughout" relationship is used to automatically indicate that an activity is to be started whenever a specific state is entered and is to be automatically stopped when the state is exited. An example is provided in Figure A-9, where the CONT_CHG activity (on the activity chart) is seen to be performed throughout the DO_CHG_PGS state (in the statechart). Another example, in Figure A-14, shows that this type of correspondence can also occur at lower levels of detail. The INIT_REVIEW activity (a subactivity of REVIEW—see Figure A-13) is performed throughout state INITIAL_REVIEW (a substate of REVIEW_MODS—see Figure A-6).

4.2.4. Structural Viewpoint — Module Charts

Module charts provide the structural description of the models; i.e., they describe who performs tasks and where they are performed. We use these charts to describe the physical aspects of the process, reflecting how it is implemented. Module charts have two major types of components: modules, and possible information flow lines. Modules represent the organizational units or individuals who perform the activities depicted in the activity chart.

There are three types of modules corresponding to three activity chart components. Execution modules show the implementation of normal activities. Environment modules correspond directly to external activities. Storage modules show the implementation location of data stores. Only the execution modules can be decomposed.

Our module chart is presented in Figure A-15. Environment modules, which are shown on the top and right-hand sides in this figure, correspond directly to external activities on the activity chart. The central portion of the diagram contains the execution modules. They are organized in a strictly hierarchical fashion that has been selected to depict the relevant portions of the organization structure at OO-ALC. For example, the MM box stands for the Directorate of Materiel Management, whose office symbol is MM. One of the suborganizations within MM is MME, the Engineering Division. Two of its subunits are MMEC (Aircraft Computer Resources Branch) and MMEDT (Operations and Support Branch, TO Section). The details of organization inside MME are shown in Figure A-16, where MMEDT is resolved into three of its subunits and a storage module labeled TO_LIBRARY. The storage module implements the data store TO_LIBS on the activity chart, as described through the textual forms facility. If one wishes to obtain more traditional views of organizational structure in the form of organization charts or printed tree-structure outlines, these are available from the system and illustrated in Figures A-17 and A-18.

Possible information flow lines are represented by solid lines in the diagram. Again, these stand out more clearly on the screen because they are designated by different colors from those of the boxes. To reduce some of the clutter in Figure A-15, information flow lines leading to and from some of the environment modules have been omitted. These suppressed lines are rather tangential to the process components that most interested us. The flow line labels on the module chart are selected to indicate the physical mechanism by which the flow is implemented. Thus, we have labels such as VERBAL, E_MAIL, US_MAIL, and HAND_CARRIED.

Connections are made between the activity chart and the module chart by use of the textual forms. Each activity can be linked to the execution module that implements it. For example, three subactivities of the DRAFT_MODS activity are linked to their corresponding modules in Figure A-19. Furthermore, each information flow on the module chart (depicting a communication channel as it is implemented) can be linked to the information flow(s) (from the activity chart) that it carries. An example is provided in Figure A-20.

4.2.5. Analysis and Simulation Capabilities

In addition to providing mechanisms for representing software process models, STATEMATE provides a number of powerful capabilities in the areas of analysis and simulation. Since we found these to be important components of a modeling approach, our experiences with them will be summarized in this section.

A few examples of the analysis capabilities will be presented here; full details on all available tests are provided in [6]. Some analysis is performed automatically while the model builder enters the specifications. For example, the system ensures that events, conditions,

data items, etc., have been defined before they can be referenced. In addition, syntax analysis is performed on event expressions, condition expressions, data expressions, and so forth, before STATEMATE accepts them as part of the model.

A wide variety of explicit tests are provided to check the model for completeness and consistency. These tests include checking for missing sources or targets of information flows and state transitions, states without incoming transitions, and activities without corresponding implementation modules. It is notable that many other design and analysis toolkits need to provide analysis facilities to ensure the consistency between different levels of abstraction; for instance, checking that the data flows leading into and out of a specific activity box on a DFD match with those shown in the decomposition of that activity, which is usually represented in a different diagram. The STATEMATE approach of using nesting within one diagram to represent decomposition, rather than using separate diagrams, eliminates the possibility of inconsistencies in levels of abstraction.

Several capabilities for deeper analysis are also provided. Some of these can be performed statically, such as checking for loops in definitions and checking the statechart for unreachable states. Other analyses are performed dynamically, meaning that STATEMATE automatically runs background dynamic simulations in order to accomplish the analysis. These include checking the statechart for reachability using the full transition labels and checking for deadlocks or nondeterminism conditions. The combination of all these analysis capabilities provides a powerful suite for ensuring the consistency, completeness, and correctness of a model; we have found these capabilities to be extremely useful.

STATEMATE also provides dynamic simulation capabilities, which are directly integrated with the model representation. The simulation works with the behavioral and functional descriptions of the model. One can begin a simulation from any valid state of the process model. Through a simulation control window and menu, one can specify changes to conditions and data values, generate events, and so forth, to emulate external influences or internal changes. After specifying these, one instructs the system to execute another simulation step, which allows the simulated process to progress according to the behavioral specification in the model. Upon the completion of that simulation step, one can examine the effects of that step, again specify changes, and continue the simulation.

When running in interactive simulation mode, STATEMATE provides a highly visual sense of the simulation progress by animating the statechart and activity chart. The state transition last taken and the current state of the system are highlighted by color on the statechart; similarly, any currently active activities are highlighted in color on the activity chart. One can also suppress the animation, or run simulations in batch mode under preprogrammed control. In all cases, a full trace of the simulation session is recorded, and any deadlock or nondeterminism situations are detected and reported.

We have extensively used the simulation capabilities in debugging our model and in validating that the formal model behaves as we understand the real process to behave. Full computational capabilities are provided for tracking quantities of interest, such as number of

TOs completing each activity at any point in the simulation; and these can be displayed interactively or written to an output file. In addition, STATEMATE now provides facilities to perform full-fledged discrete event simulations, including (1) the ability to simulate time, (2) the ability to schedule events and actions at user-specified future simulated times, and (3) the ability of the simulation to move itself forward in simulated time, carry out the next scheduled actions and events, and react to them. Unfortunately, we were not able to use this last set of features because the STATEMATE release containing them arrived too near the end of our project to allow us to work with it. Nevertheless, the many simulation capabilities we did use proved to be valuable in validating the model.

Finally, STATEMATE also provides a wide variety of reporting and inquiry capabilities. The reporting capabilities allow one to generate a wide variety of plots and reports on dictionary elements, model structure, summaries of component interactions, etc. The query capabilities are extensive, and allow one to interactively examine aspects of model structure such as decomposition paths and connections between the three perspectives. As examples, it is a simple matter to determine all the activities for which a given organizational subunit is responsible, or the communication channel used to actually implement a given information flow on the activity chart. (Again, full details on these facilities are available in [6].)

5. Results of Modeling

5.1. Enhanced Understanding of the Process

Our efforts at modeling the TO modification process led to a number of important results. First, those involved in executing and managing the process gained a substantial increase in understanding. As has been seen, several different organizational subunits are involved in various stages of the TO modification process. Not surprisingly, the view of most of these subunits was rather parochial, focusing on a specific subtask with little appreciation for overall implications. Such situations frequently lead to suboptimization with respect to overall organizational goals.

In the course of our interviews, we gained an increased appreciation of the overall goals of the process, as well as a recognition of the effect of regulations and standards on the process. As this information was communicated to the relevant personnel, we found that attitudinal changes occurred. A common understanding of the overall process and the role each group plays in its successful completion leads to increased goal congruence among those involved. It also opens the possibility of cooperating to establish process improvements, whether these are technological or manual changes.

We also found that graphical representations of the process were far more effective vehicles for communication than narrative presentations. Rapidly building a common base of understanding seems crucial in arriving at a point where fruitful discussions can occur regarding issues such as the impact of new technology, process streamlining, and effects of regulations. In short, the value of enhanced understanding of the process cannot be overstated.

5.2. Recommendations for Changes to Methods and Procedures

As our model developed, we were able to undertake an analysis of the process it represented. This analysis is reported in detail in [1]. Some of the most pervasive issues were broad in nature, regarding the relationship between the process we were examining and the broader context in which it operated. Others were more local, relating specifically to the process being examined. We directly addressed the latter category of changes in a series of recommendations. Nevertheless, we also believed that some progress could be made against the broader issues by procedural and technological changes to the TO modification process.

Our analysis led to a series of recommendations for changes to methods and procedures currently employed in the TO modification process. These recommendations could be implemented without adding any new technology; and these changes alone could have a substantial positive impact on streamlining the process and improving its timeliness. Furthermore, many of these changes would still be quite useful in conjunction with technological enhancements. These recommendations were oriented toward three major objectives:

- Eliminate delays in task initiation.
- Introduce parallelism into the process flow.
- Enhance coordination and communication in order to reduce surprises.

The detailed series of recommendations are documented in [1].

5.3. Targets for Application of Technology

Our analysis also led to the identification of specific activities as targets for the application of technology. The target list formed the basis of a pilot study applying advanced document production technology to the TO modification process. In addition, the process model provided a framework for analyzing the benefits and costs of this technology. We focused our technology efforts on the process steps preceding printing of the change documents. The application of advanced technology to the processes of identifying potential changes, reviewing them, and producing a master copy for printing appears to be highly beneficial.

The details of the technology employed in our pilot study are discussed in [2], so only a brief overview will be presented here. We utilized a state-of-the-art document production system, which included features such as:

- Full WYSIWYG (what you see is what you get) capabilities.
- Use of powerful search and replace capabilities.
- Automated referencing of figures, tables, sections, etc.
- Annotation capabilities to provide an audit trail, explain rationale for changes, and indicate approval or feedback.
- Version control including non-destructive editing and parallel development of versions.
- Use of a common element library, via inclusion by reference, to drastically reduce repetition of work.

The incorporation of this technology into the TO modification process results in a number of changes to the process description. For example, the step of preparing and issuing interim operational supplements can be completely eliminated because of the tremendous reduction in time needed to prepare a master copy of TO changes. With the new technology, formal printed op sups can be prepared fast enough to obviate the need for the interims.

The powerful non-destructive editing and version control features allow the work to be performed with greater parallelism than the manual approach allowed. Many changes can be

made and approved prior to flight testing, such as changes to part numbers and OFP identifiers. Thus, the low-level details of how individuals actually execute tasks will change with this new technology. In addition, the monolithic transfer of all TOs between the first few process steps can be broken down into individual TOs or even smaller units.

We have also demonstrated that the users can participate electronically in the review process. We have asked test pilots to review the proposed changes to the operational TOs, a valuable addition to the review step.

The above examples are representative of the changes to the process suggested by examination of our model of the current process and incorporation of the experience gained through our pilot study. All of these changes can be effectively communicated by presenting a process model of the proposed future "to-be" process. Using the STATEMATE approach, these examples result in changes to the model such as these:

- The activity and state corresponding to interim TO production and distribution are eliminated, along with the data flows and interim recipients on the activity chart and module chart.
- The increased parallelism in the early steps is depicted on the statechart.
- The involvement of pilots in the review process is depicted by adding them to the module chart and to the connection between the module and activity charts.

When the new process descriptions are then compared side-by-side with the as-is model, differences are readily apparent.

In summary, our software process modeling efforts have proven to be an important aspect of the PDSS Information Management Project. They led to increased understanding of the process by participants and managers; and they formed the basis for an analysis of the process, which led to a series of recommendations for procedural and technological changes to enhance the process.

6. Lessons Learned from Modeling

6.1. General Lessons

The first general lesson is that real-world organizational processes can be surprisingly complex; this was certainly the case for the TO modification process. As a result, it requires considerable effort to weave together the various components of the process into a cohesive whole. The information-gathering activity that is fundamental to software process modeling is an iterative process requiring multiple rounds of interviews with numerous individuals involved in the actual process steps. These rounds of interviews are necessary because of the following:

- The need to move down to additional levels of detail and/or up to higher levels of abstraction.
- The need to validate that the information gained on the last round of interviews has been faithfully represented in the model.
- The need to reconcile conflicting information obtained from different sources.
- The fact that the interviewees' understanding of the process will probably be enhanced as these meetings unfold, enabling them to better elucidate desired information in subsequent meetings.

We also discovered that interviewees sometimes left out aspects of the process in their descriptions. There are probably a number of reasons for this, some positively motivated and some not. Nevertheless, it appears to be important to cover the same ground with several individuals whenever possible to help ensure accuracy.

Furthermore, we observed that our particular modeling task was complicated by the fact that the process we were examining had not been executed repeatedly. It would appear to be substantially easier to model a process that has reached a degree of stability by having been repeated a number of times. Of course, many processes needing improvement are those which lack a history of successful application.

Finally, we have a substantial appreciation for the importance of gaining an understanding of the purposes, goals, and objectives of the overall process, as well as the major subactivities within the process being modeled. It is important that a consensus understanding be reached by the process participants. This promotes goal congruence and also makes it possible to see how some process components may be counterproductive to overall goals. In addition, it enables one to realistically consider modifications to the process to make it more effective and efficient.

As an example, suppose that we had not understood why interim TOs are issued in the process we studied. Lacking an appreciation of the reason, we probably would have assumed that this step was necessary to the actual purpose of the overall process. Then, we would not have recommended eliminating it when using advanced document production technology. However, because we had learned that this step was present only because it

previously took so long to prepare and print the formal change documents, we knew that its elimination would be a welcome enhancement, as long as formal documents could be quickly prepared. Thus, knowledge of the purpose of the process and the rationale for the various steps are enormously important for successful process improvement efforts.

6.2. Requirements for a Modeling Approach

We have formulated a set of requirements for an ideal approach to software process modeling. Actually, these requirements can be more appropriately viewed as highly desirable characteristics of a suitable modeling methodology, since the availability of even a portion of these characteristics is expected to yield substantial benefits. The composition of this list is based on a combination of factors, including our experiences with those capabilities we had an opportunity to use, our identification of capabilities that we felt were notably absent, and a view toward the future to which software process modeling could lead. These desirable characteristics are described below.

1. Use a highly visual approach to information representation, such as diagrams. Well-chosen graphical representations can readily convey considerable amounts of information. We found such depictions to be very useful in facilitating communication.
2. Enable compendious descriptions. The most useful descriptions are comprehensive in scope yet concise in presentation, so that complex aspects may be represented relatively easily.
3. Support multiple, complementary views of the process. We have identified four viewpoints, or perspectives, which are quite useful:

    - functional
    - behavioral
    - implementation
    - conceptual data modeling

    The idea of perspectives is analogous to the perspectives presented in an engineering drawing; for example, a top view, a front view, and a side view. These viewpoints are distinct, yet they are interrelated because they describe the same reality from a different point of view. In software process modeling, the reality being described is that of the real-world process, and the perspectives focus on different, but closely related, aspects of that process.

    The functional viewpoint describes what activities are being performed. The data flow diagram presented earlier is an example of a functional description; it focuses on what the main activities are and what data flows between them.

    The behavioral viewpoint describes when and how these activities are accomplished. The formalism for this perspective must be capable of representing feedback loops, iteration, complex decision-making conditions, entry criteria (trigger conditions), exit criteria, precedence relationships, etc. In addition, it should readily represent and convey two levels of parallelism. At the lower level are individual software objects; for example, both unit testing/debugging

and documentation revision may occur simultaneously for a single module. The higher level of parallelism occurs across objects; for instance, some modules may be in test while others are still in coding. It must also be possible to handle synchronization with other (groups of) objects as, for example, when all components must have passed unit testing before integration testing can begin.

Next, the implementation viewpoint describes who and where the activities are implemented. It connects the activities with the organizational subunits performing the work and may also be used to describe communication channels (e.g., e-mail, verbal communication, or written document).

Fourth, the conceptual data modeling perspective represents an abstract, global view of relevant data on the software objects being produced as well as on the process itself. This viewpoint is the avenue through which one connects process and product metrics to the process representation; it also supports other quantitative efforts, such as simulation.

Finally, the approach should allow flexibility in the order in which these viewpoints are developed.

4. Support multiple levels of abstraction (e.g., hierarchical decomposition) for each viewpoint. This gives a perspective on how a subprocess fits in context and also allows one to delve into its details. In other words, it allows one to get a big picture of the process, as well as its low level details and the levels in between. It must also be possible to stop the refinement at any point and still have a meaningful model. As Lehman points out, this is likely to occur at creative points in the process, such as "devise alternatives," which humans will have to carry out [8]. Finally, various approaches to abstraction should be supported, e.g., top-down, bottom-up, and middle-out.

   It must also be possible to drive different portions of the model to varying levels of detail. For example, an activity of little interest may not be decomposed further, while a crucial one may undergo several levels of resolution into detail.

5. Offer a formally defined syntax and semantics, so that the constructs are computable. This allows a process description to be parsed and semantically analyzed. It also means that diagnostic information can be automatically derived and reported to the originator of the description [9]. In addition, this opens the possibility of automatically simulating or executing the description.

6. Provide comprehensive analysis capabilities. These would involve tests in categories such as the following:

   - Consistency: test for balance of information flows between levels of abstraction, consistency between multiple viewpoints, conflicting trigger conditions, etc.
   - Completeness: check for errors such as missing sources or targets for information flows, state transitions, etc., data stores without output flows, states without incoming transitions.
   - Correctness: check for problems such as syntax errors, unreachable paths, loops in definitions, deadlocks or nondeterminism in the behavioral specification.

These sorts of analysis capabilities are extremely useful during model building. Anomalies detected by analysis routines may have a number of causes:

- The anomaly may be a simple error in the formal description of the model, that is, in the translation of the model builder's correct understanding to the formalism of the model.
- It may indicate that the model builder does not have an accurate understanding of some aspect(s) of the real-world process, meaning that additional information gathering and clarification is required.
- It may indicate that there are anomalies in the real-world process, such as nondeterminism. These anomalies are probably unknown to the process participants, and the existence of such anomalies is valuable feedback to them.

7. Facilitate the simulation of process behavior directly from the description. We favor the provision of interactive simulation capabilities that animate the model description for clarity. In addition, for more involved studies we recommend batch simulation capabilities that save a detailed trace. A qualitative simulation capability which illustrates the behavior and reactions of the model to changing conditions and events, is invaluable in checking the validity of formal descriptions and their robustness to changing conditions. This is an important complement to the analysis capabilities when debugging a model. Simulations should also support the quantitative analysis of attributes of interest, e.g., time-to-completion, manpower requirements, and quality measures. These capabilities would be a powerful mechanism for thoroughly examining contemplated process modifications and alternatives before using them in practice.
8. Readily support the creation and management of variants, versions, and reusable components of process descriptions. It should be easy to reuse portions of descriptions and easy to identify differences between descriptions. We envision building libraries of reusable process components which can be readily selected and inserted during model building. As an example, technical reviews occur repeatedly in any well-managed software development process; a common review component could be defined and used repeatedly as required throughout a model. It should be easy to manage variants of a model, as might occur when modeling different but very similar processes. Versions of models should also be managed conveniently; these would occur, for example, when developing an as-is model and then a proposed future to-be version of the process. Facilities should be provided to highlight differences, thus assisting comparison of model descriptions. Finally, it should be possible to record deliberate tailoring decisions into the model directly. (This illustrates that these decisions are an important part of the process and helps ensure that they will be reliably repeated in the future, when the model is used as a guideline for process execution.)
9. Support the representation and analysis of constraints on the process, such as regulations, standards, and so on. In the defense community, there are a number of rules specifying required documents and formats, review points, etc. It would be useful to reflect these constraints somehow within the modeling formalism. It would then be highly desirable to be able to automatically

analyze a process model to determine the extent to which it complies with the constraints.

10. Allow the representation of purposes, goals, and rationales for process components and the overall process. As discussed above, such knowledge is crucial to successful process improvement efforts. It would be convenient if this knowledge could be incorporated into the model, even if it is in the form of unprocessed narrative commentary.
11. Integrate easily with other approaches which may be deemed useful: for example, PERT or CPM for critical path analysis, or entity-relationship modeling as a conceptual data model.
12. Take an active role in process execution. Such capabilities might include automatically recording the steps taken during actual execution of the subject process, along with data about the process and the objects manipulated. The model might also play an assistant role by providing guidance in executing the process based upon the descriptions. Clearly, these capabilities are oriented more toward the future, rather than the near term.
13. Offer automated tools supporting the approach. Certainly, the availability of automated support tools dramatically enhances the practical applicability of a modeling approach. Nevertheless, we do not include typical tool evaluation criteria (e.g., vendor support and user interface characteristics) in this list, since the focus here is on functionality and methodology.

7. Conclusion and Future Directions

This report has presented a description of our views on software process modeling. We have attempted to stress the importance of this emerging discipline both in general and specifically through a description of our experiences with modeling on a recent project. We have presented and discussed examples of models in the context of the approach to modeling that we have taken utilizing STATEMATE. Finally, we covered a series of lessons learned from our modeling experience and presented a list of major characteristics which we believe are important in an approach suitable for software process modeling.

In our view, there are several additional efforts that would be highly beneficial to the near-term development of software process modeling. One is to investigate the set of existing methodologies and toolkits that may be applied to software process modeling. The above requirements list would be a major starting point in such an investigation. Promising approaches should be tried on a comprehensive trial modeling problem to determine suitability. Another valuable contribution would be to examine the quantitative simulation capabilities of approaches like STATEMATE and to demonstrate these capabilities in a realistic model. Finally, we recommend gaining additional experience by modeling more aspects of the software life cycle process. It would also be instructive to model a few variants of a process and then synthesize a general, comprehensive model from these separate models.

In conclusion, we have gained valuable experience with the emerging discipline of software process modeling. It is the authors' judgment that modeling approaches such as the STATEMATE system can be effectively used for modeling software processes. We look forward to carrying out some of the future work indicated above and contributing to the maturation of software process modeling as a technique for aiding in the evolution of software processes.

Appendix A: Figures

Activity 4.0 - Print Documents

ID Number: 4.0

Activity Name: Print Documents

Activity Description:
The purpose of this activity is to print the required number of copies of each op sup or group of TO change pages. Three options are available for the actual printing of each job:

- printing by shop on the base
- printing by local contractor (19 printers currently on direct deal contracts through GPO)
- work through GPO regional office in Denver

The last option is used only when the job cannot be accomplished by the other means. Choosing which of the first two approaches to use involves considering the workload in the base printing facility and the rules of the Joint Committee on Printing (JCP - a Congressional committee). JCP rules specify situations when printing must be accomplished through GPO, based on numbers of copies produced, total pages to be printed, and the type of printing equipment on base.

The time limit for printing is generally 15 days. This requirement is met in most cases; if there is a delay, it is usually just a couple of days.

Office Accomplishing Work: OO-ALC/DARA

Timing:

- Commencement — This activity begins for each TO as the reproducible copy is received from MMEDT.
- Termination — It ends when the printed documents have been received and sent to the TO warehouse for distribution.

Inputs:

- Reproducible copy of each affected TO

Outputs:

- Printed copies of each affected TO

Figure A-1: Example of Structured Narrative Description

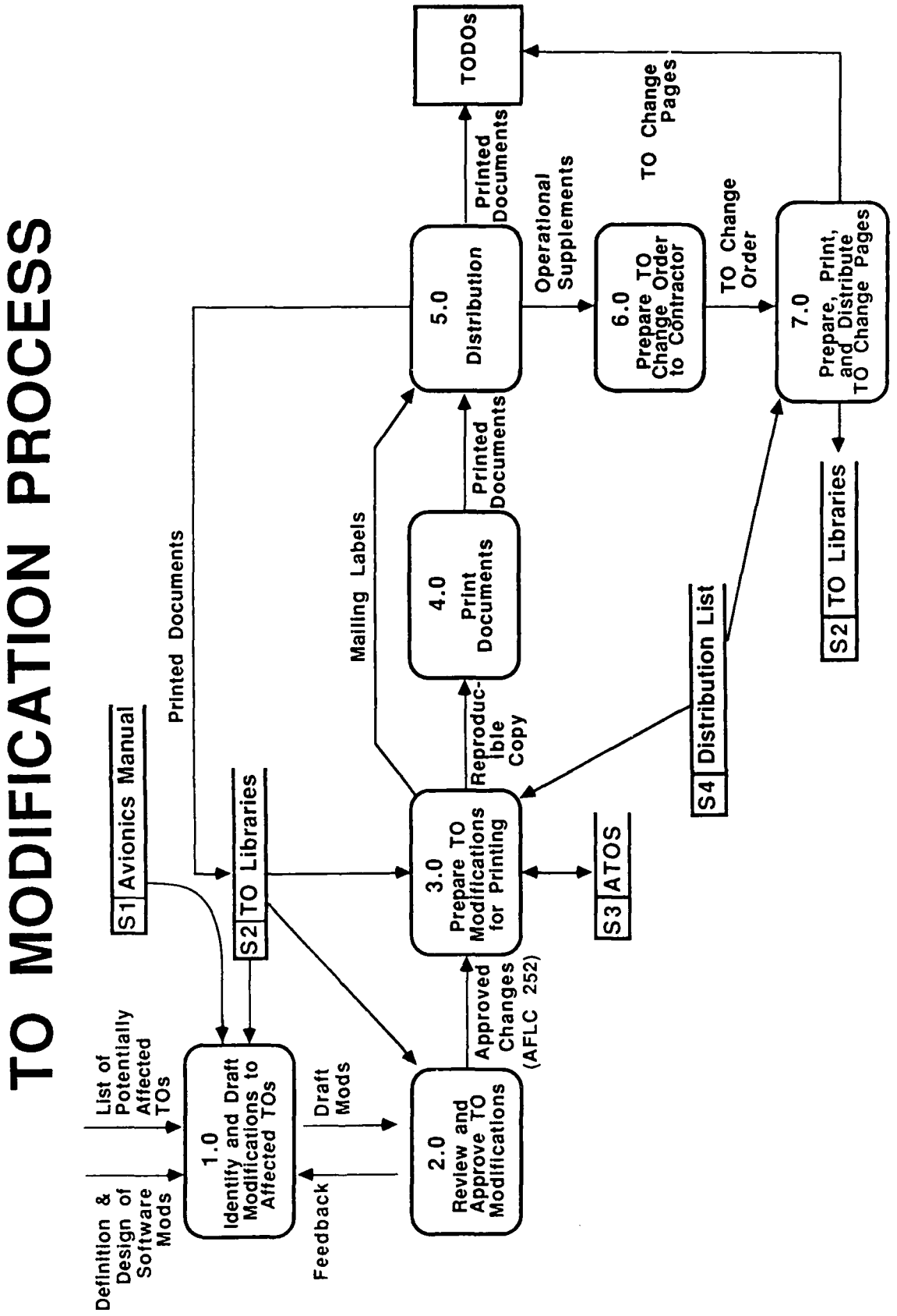


Figure A-2: Data Flow Diagram of TO Modification Process

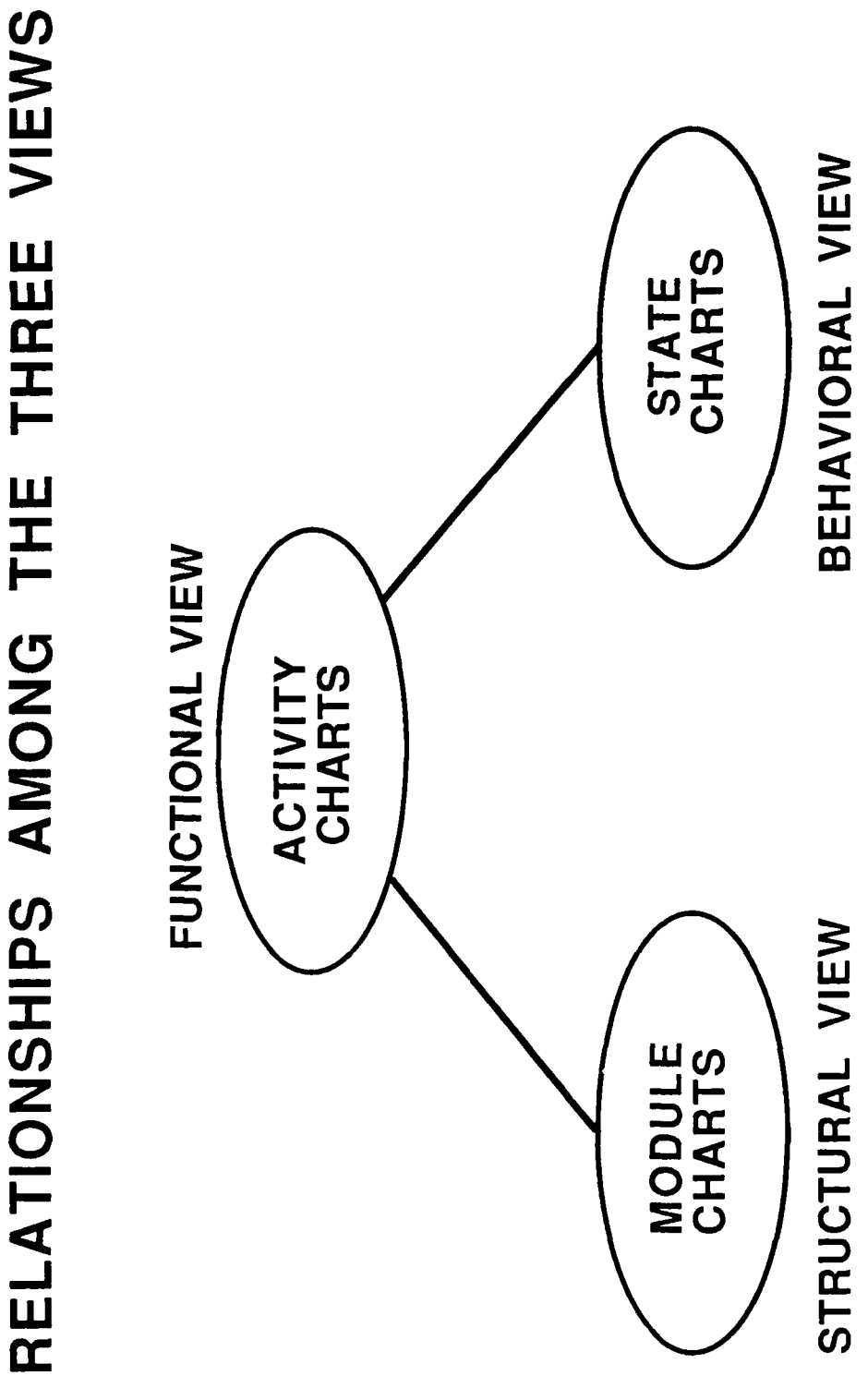


Figure A-3: Relationships Between STATEMATE Views

```
EVENT DICTIONARY
Detailed TO Process Model

NEED_REVS :   (INIT_REV_DONE or RE_REV_DONE) [REWORK_NEEDED]

description:  Revisions need to be made to TO modifications.
```

Figure A-4: Example of Event Definition

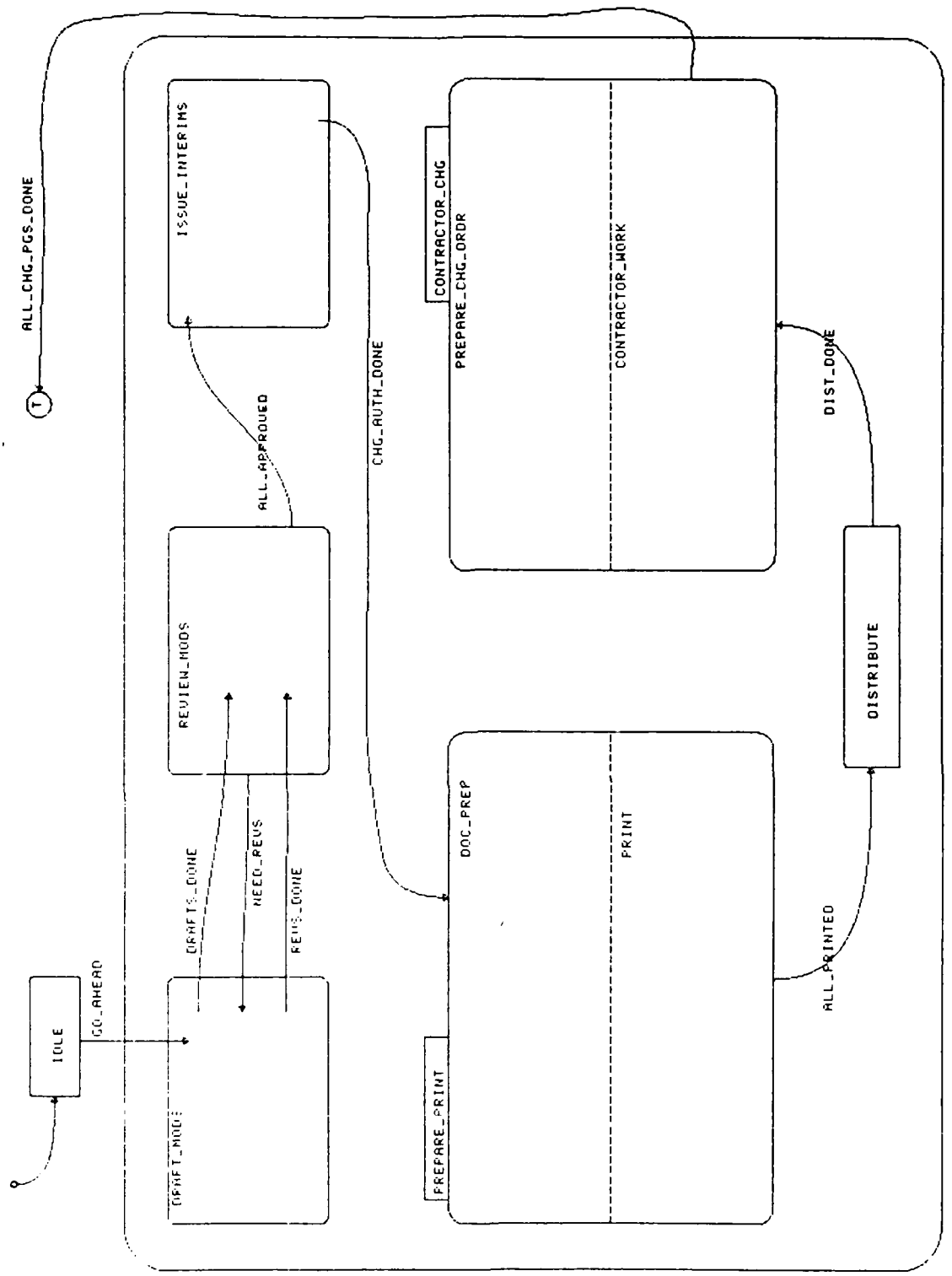


Figure A-5: Statechart - Top-Level Process

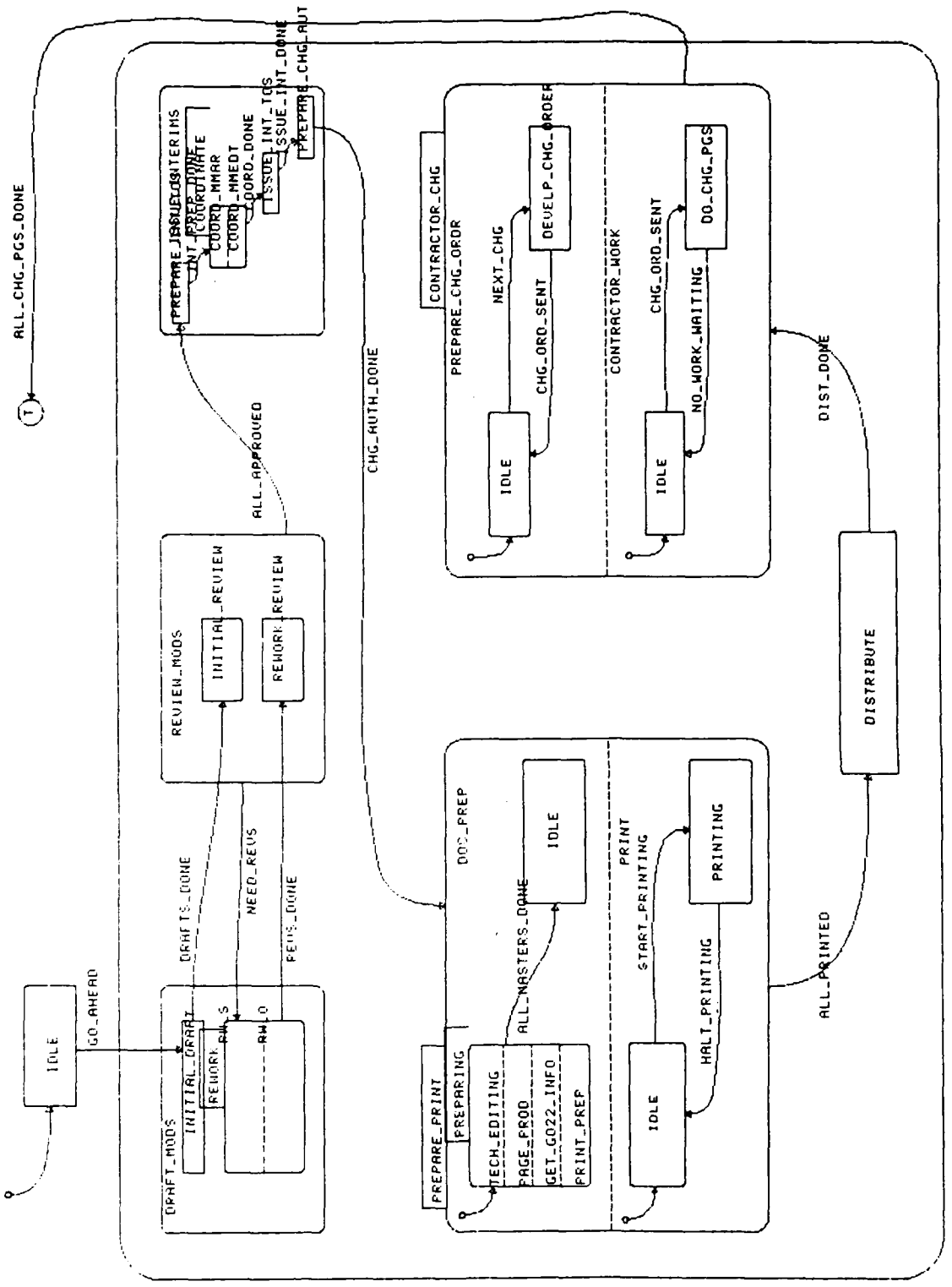


Figure A-6: Statechart - Detail-Level Process

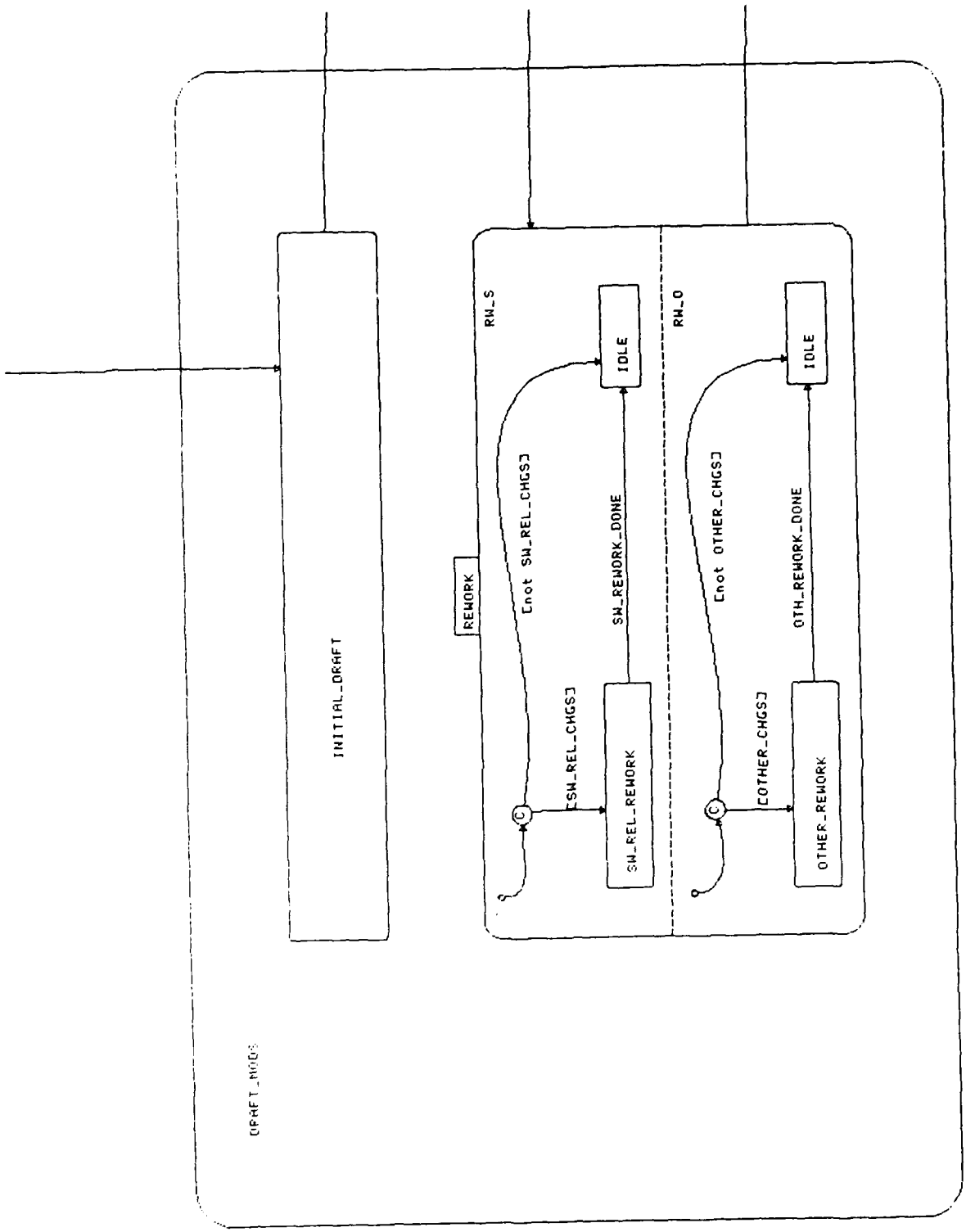


Figure A-7: Statechart - Detail of DRAFT_MODS State

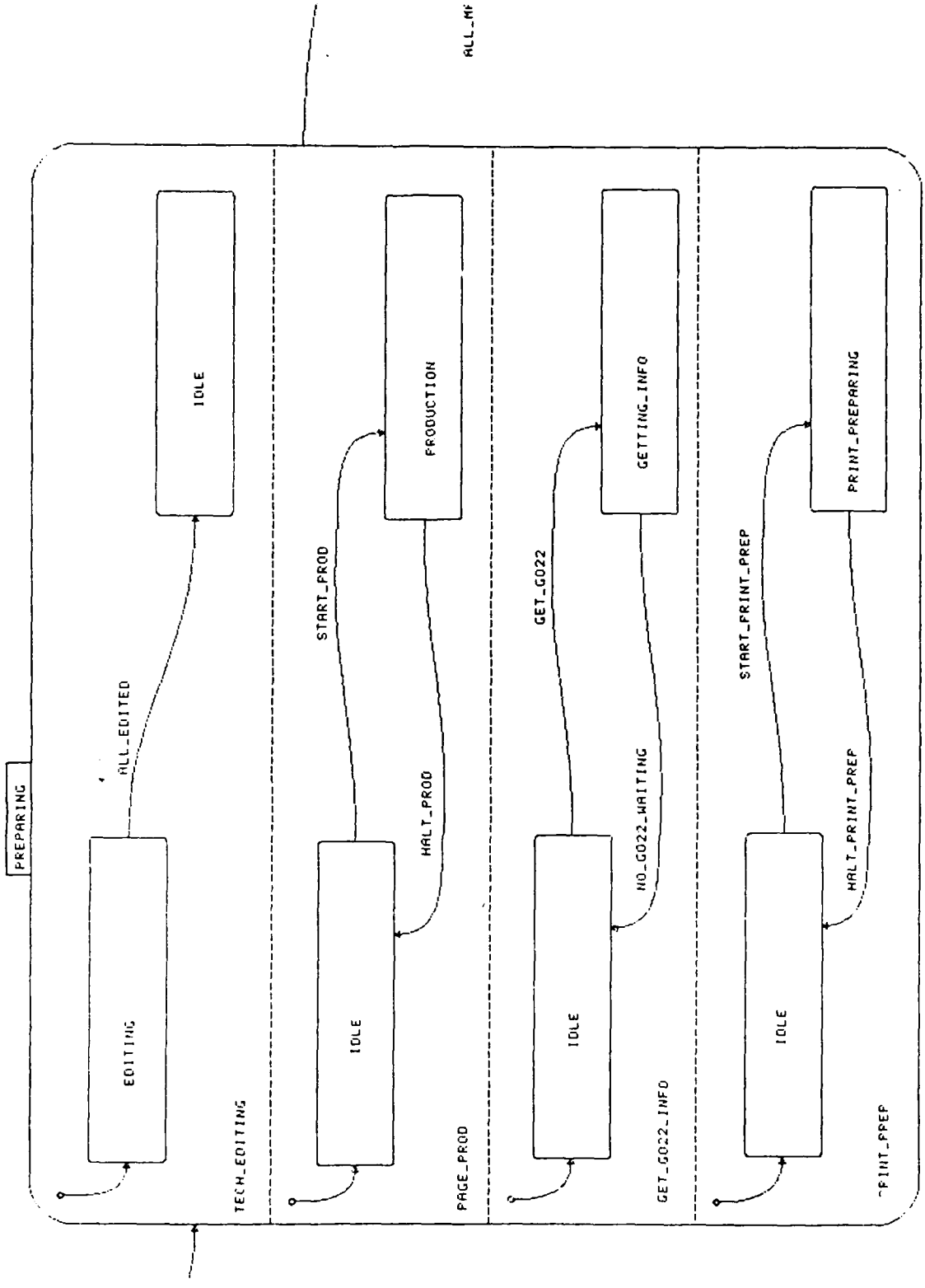


Figure A-8: Statechart - Detail of PREPARING State

```
STATE DICTIONARY
Detailed TO Process Model


DO_CHG_PGS

type:  BASIC

activities throughout:  CONT_CHG

reactions:
 on:                                    do:
A_TO_CHG_BY_CONT                        NUM_FINAL_CHGS:=NUM_FINAL_CHGS+1
A_TO_CHG_BY_CONT                        NUM_IN_CONT_Q:=NUM_IN_CONT_Q-1
CHG_ORD_SENT                            NUM_IN_CONT_Q:=NUM_IN_CONT_Q+1
ENTER                                   NUM_IN_CONT_Q:=NUM_IN_CONT_Q+1

description:  Contractor develops, prints, and distributes change pgs for TOs.
```

Figure A-9: Example of State Definition of DO_CHG_PGS

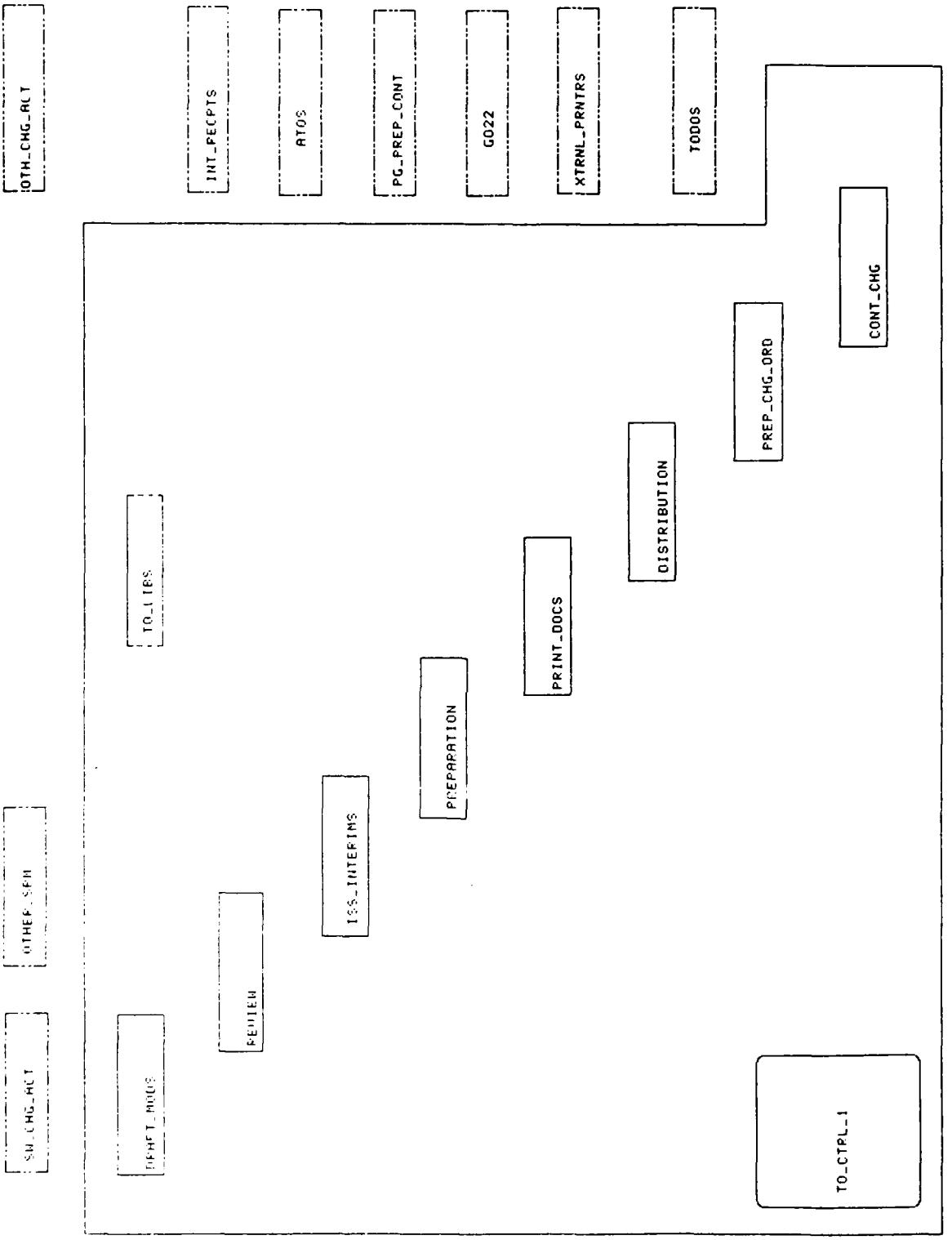


Figure A-10: Activity Chart - Full Process - No Flows

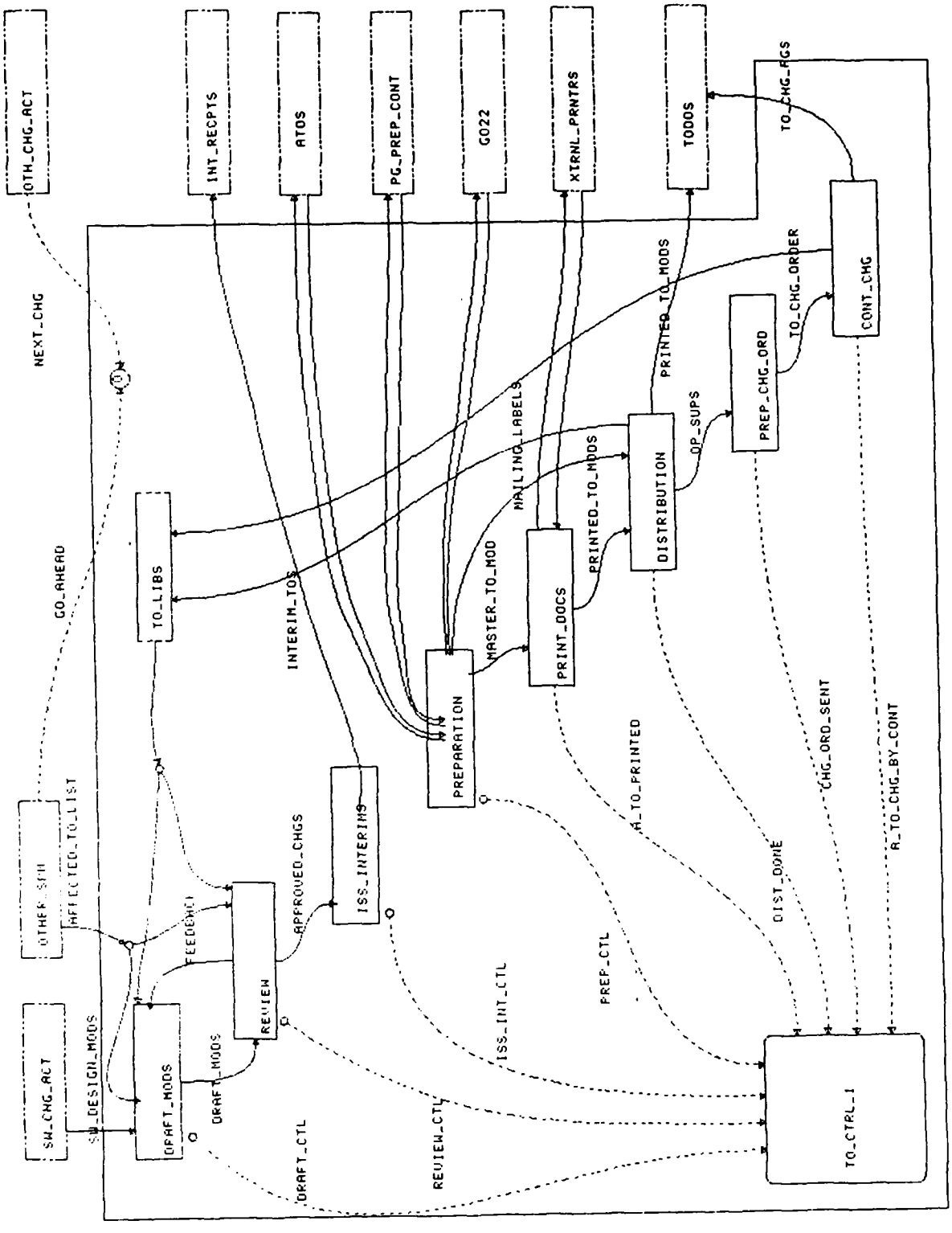


Figure A-11: Activity Chart - Full Process - With Flows

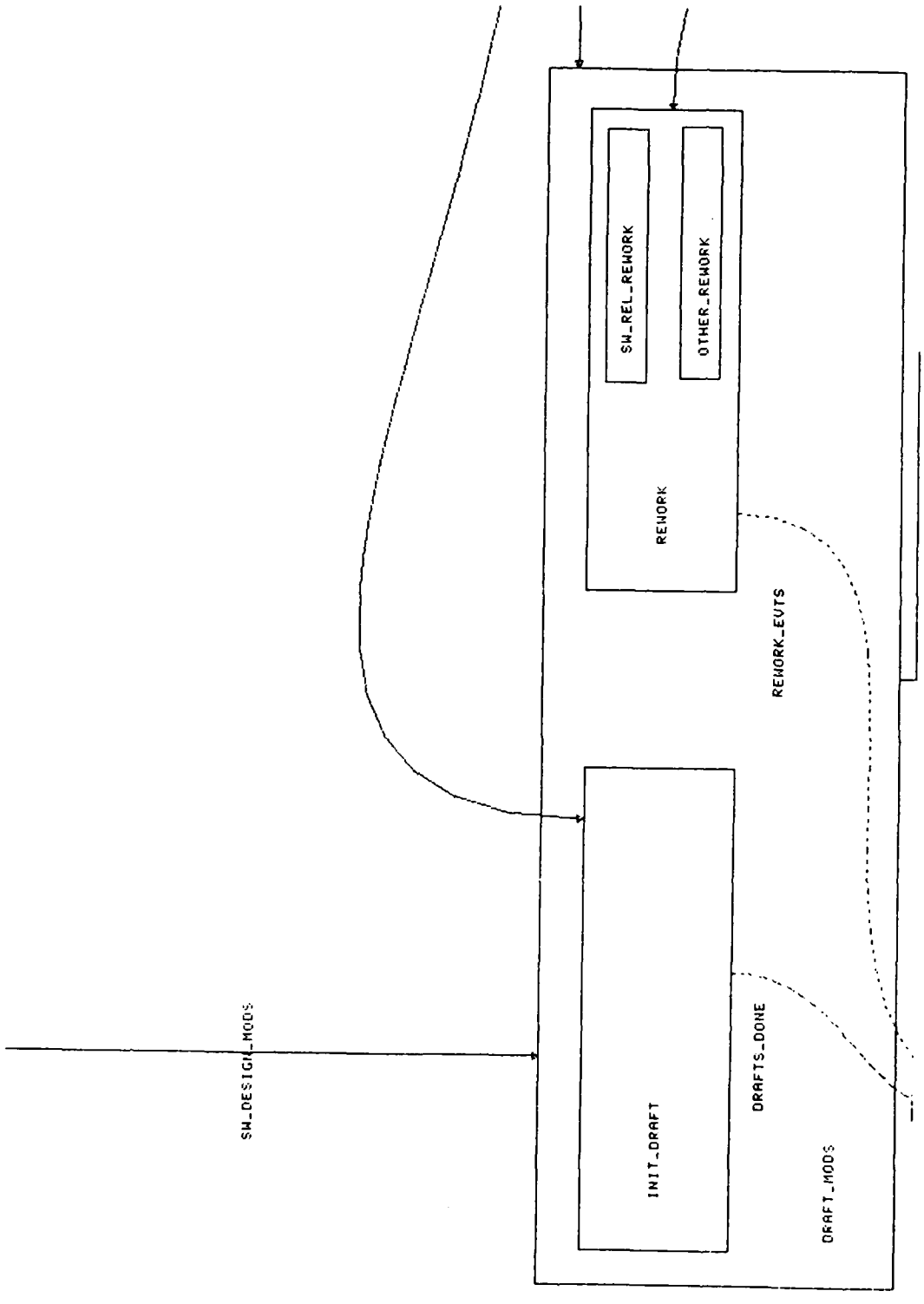


Figure A-12: Activity Chart - Detail of DRAFT_MODS Activity

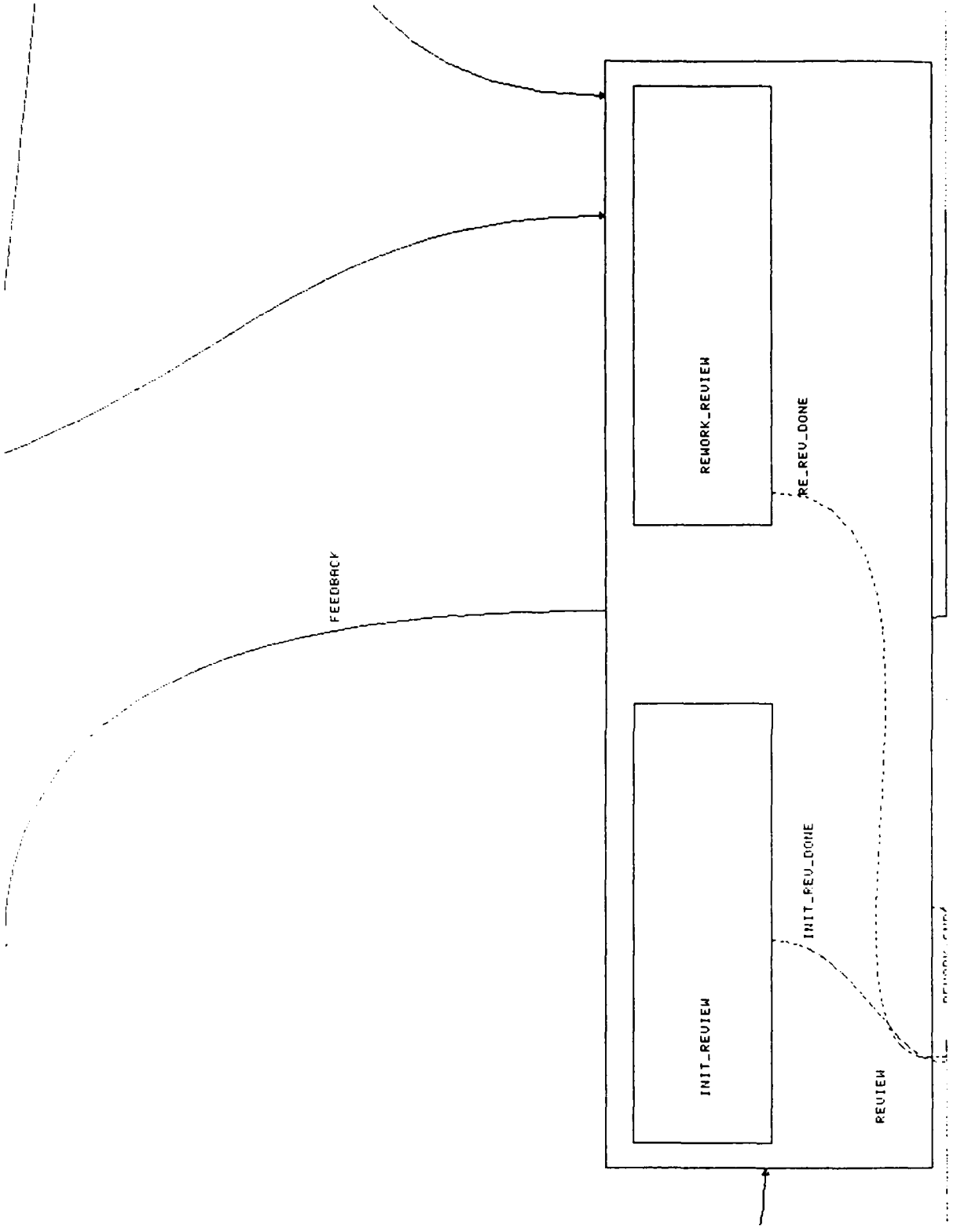


Figure A-13: Activity Chart - Detail of REVIEW Activity

```
STATE DICTIONARY
Detailed TO Process Model

INITIAL_REVIEW

type:  BASIC

activities throughout:  INIT_REVIEW

description:  Perform review of initial draft TO mods.
```

Figure A-14: Example of State Definition of INITIAL_REVIEW

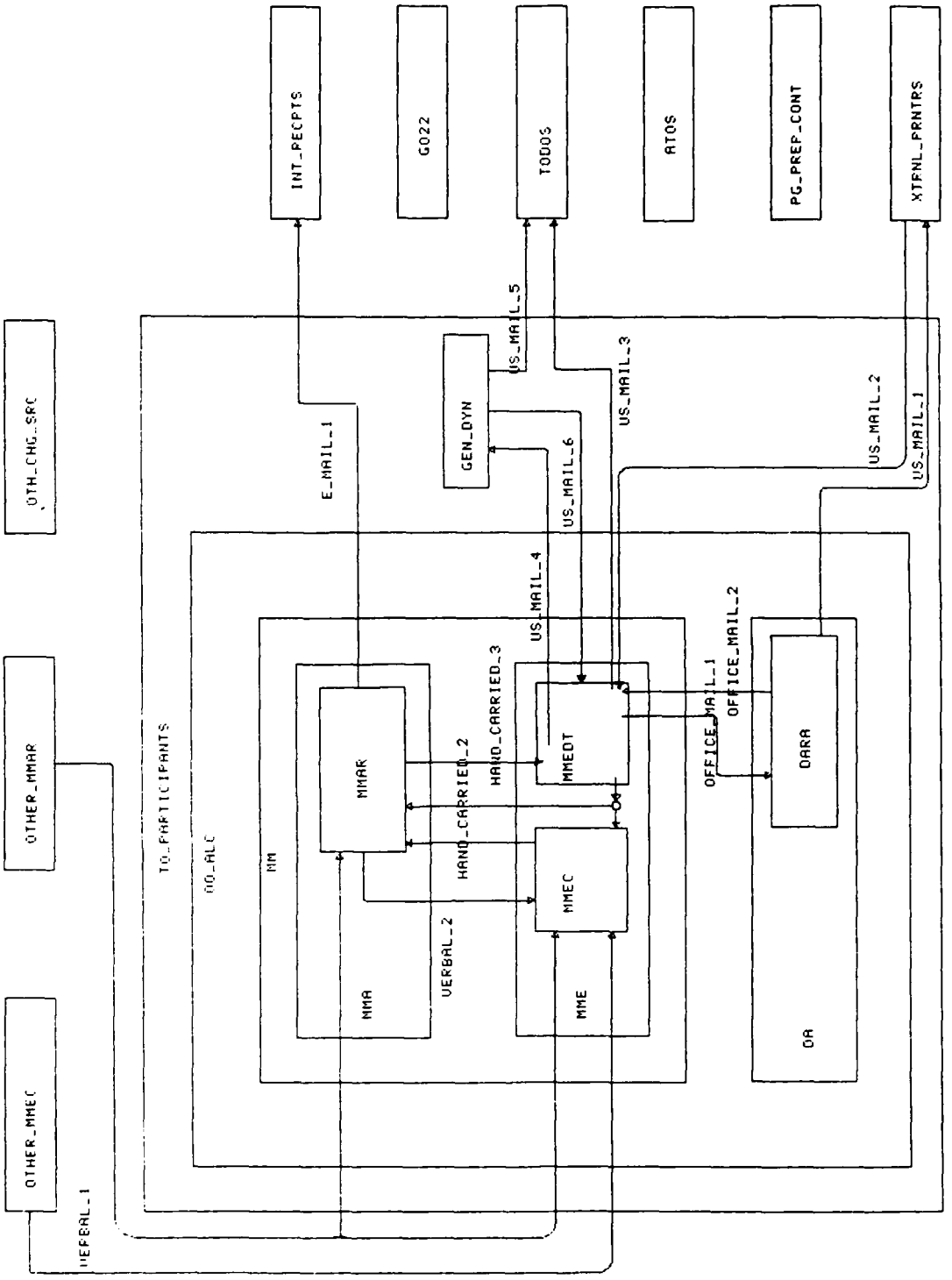


Figure A-15: Module Chart - Top-Level Process

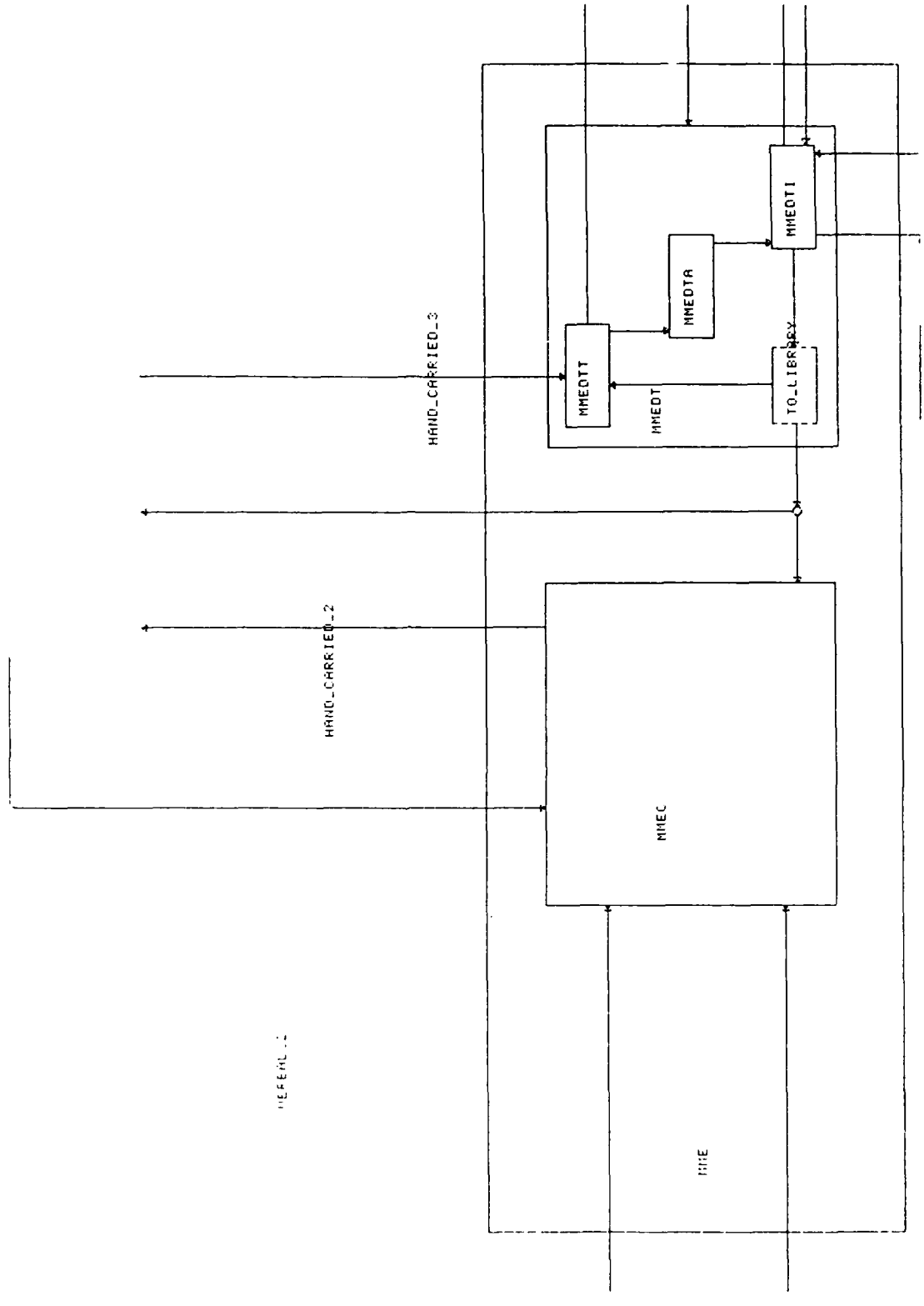


Figure A-16: Module Chart - Detail of MME Module

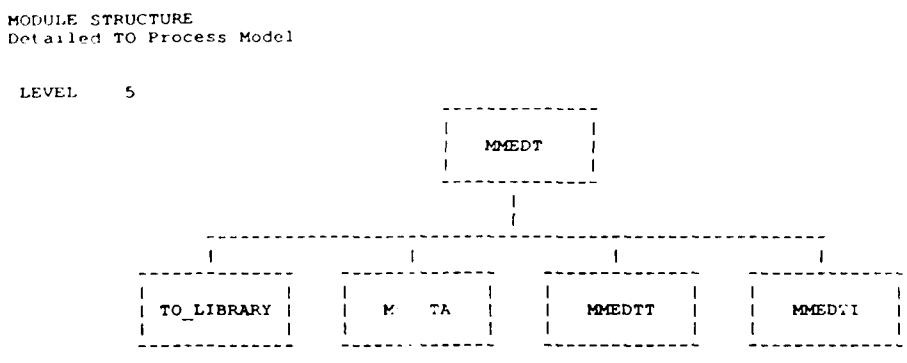


Figure A-17: Example Organization Chart

```
MODULE TREE
Detailed TO Process Model

TREE FOR TO_PARTICIPANTS
========================

TO_PARTICIPANTS

      1.  OO_ALC

          1.  DA

              1.  DARA

          2.  MM

              1.  MME

                  1.  MMEC

                  2.  MMEDT

                      1.  TO_LIBRARY

                      2.  MMEDTA

                      3.  MMEDTT

                      4.  MMEDTI

              2.  MMA

                  1.  MMAR

      2.  GEN_DYN
```

Figure A-18: Outline Form of Module Hierarchy

```
ACTIVITY DICTIONARY
Detailed TO Process Model

INIT_DRAFT

implemented by:  MMEC

description:  Perform first draft of TO modifications.

-------------------------------------------------------------------------------

SW_REL_REWORK

implemented by:  MMEC

description:  Rework drafts of TO mods especially related to SW details.

-------------------------------------------------------------------------------

OTHER_REWORK

implemented by:  MMAR

description:  Rework drafts of TO mods not especially related to SW details.
```

Figure A-19: Example of Activities Implemented by Modules

```
INFORMATION-FLOW DICTIONARY
Detailed TO Process Model


US_MAIL_3

contains:  PRINTED_TO_MODS


-------------------------------------------------.


US_MAIL_5

contains:  TO_CHG_PGS
```

Figure A-20: Example of Relationships Between Flows in Activity and Module Charts

References

[1] Hansen, Greg, Kellner, Marc, Over, James and Przybylinski, Stanley.
The Analysis of the Technical Order Production Process at Ogden Air Logistics Center and Recommendations for the Improvement of the Process.
Technical Report CMU/SEI-87-TR-12, Software Engineering Institute; Carnegie Mellon University, January, 1988.

[2] Hansen, Greg, Kellner, Marc Kellner, and Over, James.
Technology Application to Documentation Maintenance.
Technical Report, Software Engineering Institute,
Carnegie Mellon University, 1988.
forthcoming.

[3] Harel, David.
Statecharts: A Visual Formalism for Complex Systems.
Science of Computer Programming 8(3): 231-274, June 1987.

[4] Harel, David, et al.
On the Formal Semantics of Statecharts.
In *Proceedings of the 2nd IEEE Symposium on Logic in Computer Science*, pages 54-64. IEEE, 1987.

[5] *The Languages of STATEMATE*
i-Logix, Inc., 22 Third Avenue, Burlington, MA 01803, March 1987.

[6] *STATEMATE User's Manual; Kernel and Analyzer; VAXstation Version 1.2*
i-Logix, Inc., 22 Third Avenue, Burlington, MA 01803, February 1988.

[7] Joint Logistics Commanders.
Final Report of the Joint Logistics Commanders Workshop on PDSS for Mission-Critical Computer Software,
June 1984.

[8] Lehman, M. M.
Process Models, Process Programs, Programming Support.
In *Proceedings of the 9th International Conference on Software Engineering*, pages 14-16. IEEE, 1987.

[9] Osterweil, Leon.
Software Processes Are Software Too.
In *Proceedings of the 9th International Conference on Software Engineering*, pages 2-12. IEEE, 1987.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE DISTRIBUTION UNLIMITED
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-88-TR-9	5. MONITORING ORGANIZATION REPORT NUMBER(S) ESD-TR-88-010
6a. NAME OF PERFORMING ORGANIZATION SOFTWARE ENGINEERING INSTITUTE 6b. OFFICE SYMBOL (If applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI JOINT PROGRAM OFFICE
6c. ADDRESS (City, State and ZIP Code) CARNEGIE MELLON UNIVERSITY PITTSBURGH, PA 15213	7b. ADDRESS (City, State and ZIP Code) ESD/XRS1 HANSCOM AIR FORCE BASE, MA 01731
8a. NAME OF FUNDING/SPONSORING ORGANIZATION SEI JOINT PROGRAM OFFICE 8b. OFFICE SYMBOL (If applicable) SEI JPO	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962885C0003
8c. ADDRESS (City, State and ZIP Code) CARNEGIE MELLON UNIVERSITY SOFTWARE ENGINEERING INSTITUTE JPO PITTSBURGH, PA 15213	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: PROJECT NO.: N/A TASK NO.: N/A WORK UNIT NO.: N/A
11. TITLE (Include Security Classification) SOFTWARE PROCESS MODELING	
12. PERSONAL AUTHOR(S) Kellner, Marc I. and Hansen, Gregory A.	
13a. TYPE OF REPORT FINAL 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) May 1988 15. PAGE COUNT 60
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	documentation, software process, PDSS, STATEMATE, process modeling

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Software Engineering Institute (SEI), located in Pittsburgh, Pennsylvania, is a federally funded research and development center operated by Carnegie Mellon University under contract to the Department of Defense. An objective is to provide leadership in software engineering and in the transition of new software engineering technology into practice. This paper discusses a software process modeling case study conducted at the SEI.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☒	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED, UNLIMITED
22a. NAME OF RESPONSIBLE INDIVIDUAL KARL SHINGLER	22b. TELEPHONE NUMBER (Include Area Code) (412) 268-7630 22c. OFFICE SYMBOL SEI JPO